

The Cinegy Cinecoder Library

API documentation

Table of Contents

Introduction	1
Cinecoder Concepts	3
Terms and Definitions	4
COM API	5
Naming Conventions	6
Data Types	7
Basic Data Types	8
Multimedia Data Types	9
Consumers and Producers	10
ByteStream interfaces	11
MediaData Interfaces	12
The Hierarchy	13
Base Interfaces	14
Decoders and Encoders	15
De- and Multiplexers	18
The Low Latency Mode	19
Class Factory	21
Creating objects and Smart Pointers	22
Licensing	23
Cinecoder API	25
Base API	26
ICC_ElementaryDataInfo Interface	27
ICC_ElementaryStreamInfo Interface	30
ICC_StreamRecognizer Interface	32
Settings API	33
Settings elements	34

General settings properties	36
Serialization	37
XML data format	38
Schemas API	39
Creation of Schema objects	40
Schema definition	41
The Schema Sample	42
Video API	43
Interfaces	44
Structures	60
Audio API	75
ICC_AudioConsumer Interface	76
ICC_AudioProducer Interface	78
ICC_AudioDecoder Interface	81
ICC_AudioEncoder Interface	84
ICC_AudioFrameInfo Interface	86
ICC_AudioStreamInfo Interface	87
ICC_AudioEncoderSettings Interface	89
CC_AUDIO_FMT Enumeration	91
Multiplex API	93
ICC_Demultiplexer Interface	95
ICC_Multiplexer Interface	98
ICC_BaseMultiplexerPinSettings Interface	101
ICC_DemultiplexedDataCallback Interface	103
ICC_MultiplexedStreamInfo Interface	105
ICC_MultiplexedDataDescr Interface	107
ICC_BaseMultiplexerSettings Interface	109
CC_ACCESS_UNIT_DESCR Structure	111
CC_ELEMENTARY_STREAM_TYPE Enumeration	112
CC_MULTIPLEXED_STREAM_TYPE Enumeration	116
CC_MUX_OUTPUT_POLICY Enumeration	117
CC_PACKET_DESCR Structure	118
CC_PES_ID Enumeration	119
CC_PSI_TABLE_ID Enumeration	120

MPEG_SYSTEM_DESCRIPTOR_TAG Enumeration	121
--	-----

MPEG-2 codec **123**

MPEG Video	125
Types	126
Interfaces	134
MPEG Video Decoder	142
MPEG Video Encoder	146
D10/IMX Video Encoder	159
MPEG Audio	164
MPEG Audio Encoder	165
MPEG Audio Decoder	166
MPEG Multiplex	167
ICC_ProgramInfo Interface	168
ICC_SystemDescriptorsManager Interface	170
ICC_PES_Info Interface	173
ICC_TS_ProgramDescr Interface	176
ICC_SystemDescriptorsReader Interface	178
MPEG-2 Program Stream	181
MPEG-2 Transport Stream	187
MPEG-1 System Stream	194
HDV-1 Multiplexer	198
HDV-2 Multiplexer	199

MPEG-4 codec **201**

AVC/H.264 Video	203
Overview	204
Types	206
Interfaces	210
H.264 Video Decoder	216
H.264 Video Encoder	218
AVC-Intra Encoder	229
AAC Audio	234

Overview	235
Types	236
Interfaces	238
AAC Encoder	241
AAC Decoder	242
MP4 Multiplex	243
ICC_MP4_Multiplexer Interface	244
ICC_MP4_MultiplexerSettings Interface	246
ICC_MP4_MuxerPinSettings Interface	248
Additional codecs	251
AES-3 (SMPTE-302M) Audio codec	252
Samples	253
H.264 Video Decoder Sample	254

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;

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

Naming Conventions

The cinecoder interfaces have the "ICC_ " (Interface of CineCoder) prefix. Then the interface name follows (with the capitalized words), for example:

- ICC_MpegVideoEncoder
- ICC_ByteStreamConsumer

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 ByteStream.

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.

MediaData Interfaces

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

- For audio: Consumer + ICC_AudioProducer ([↗](#) see page 78)
- For video: ICC_VideoConsumer ([↗](#) see page 44) + ICC_VideoProducer ([↗](#) see page 46)

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`.

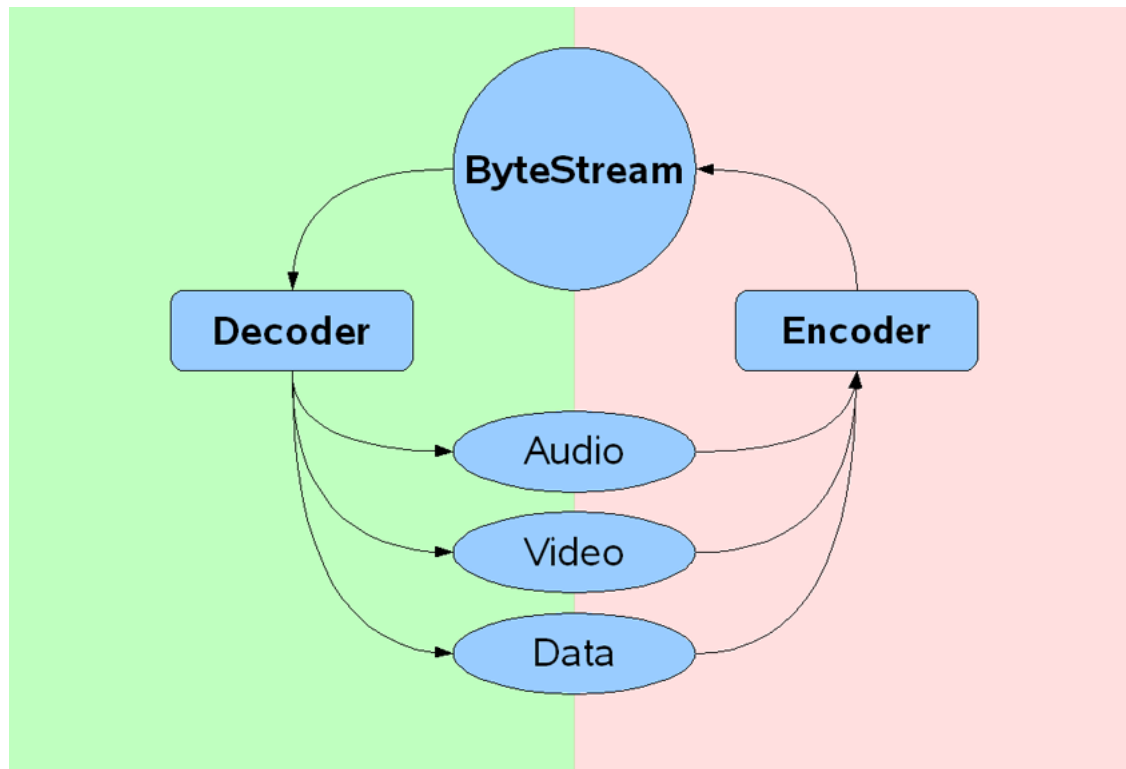
Any media data naturally have information about the time when these shall be “published”, the so-called presentation time. The `TimeBase` property sets up the scale for those time intervals, measured in Hertz.

Decoders and Encoders

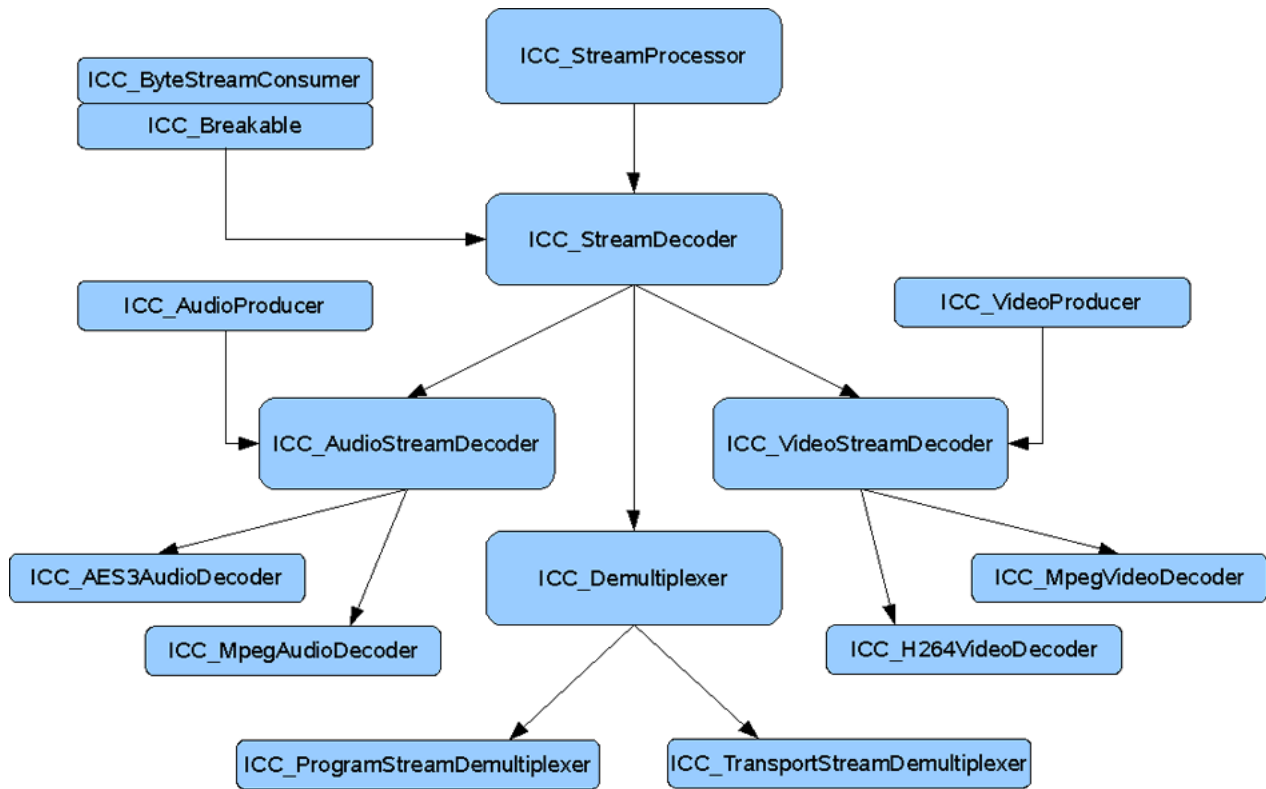
The cinecoder objects are divided into two main types in relation to the ByteStream (see page 9); these are:

- The decoders (ICC_Decoder), and
- The encoders (ICC_Encoder).

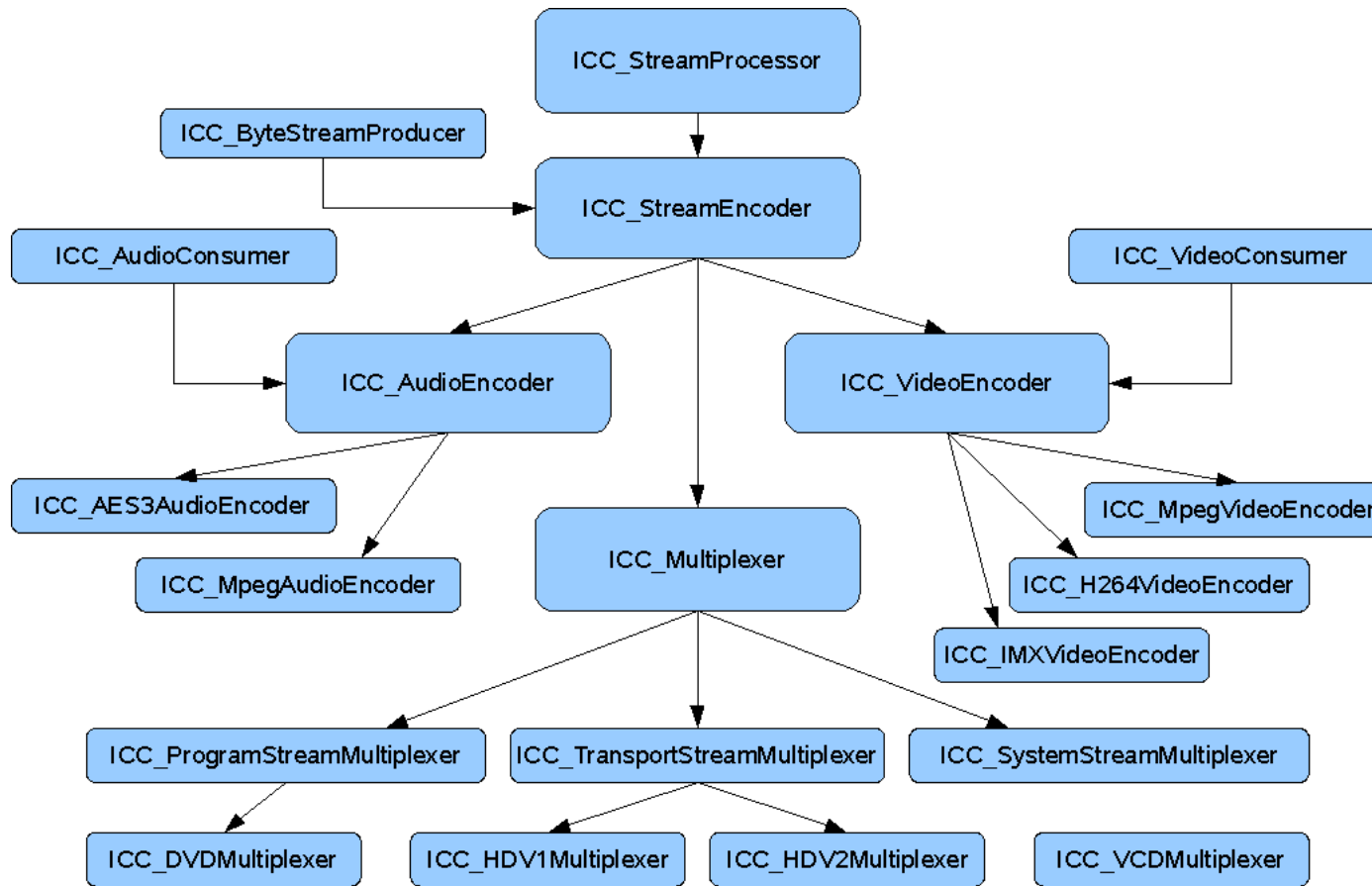
The decoders accept the ByteStream (see page 9) as the input, the encoders produce the ByteStream (see page 9). A simplified scheme is presented in the following illustration:



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 93](#)).

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 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). $Latency = Latency_{header}$

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);
```

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 API

Base API

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

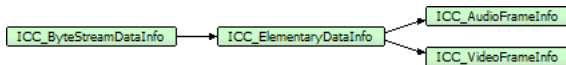
Interfaces

	Name	Description
	ICC_ElementaryDataInfo ( see page 27)	The description of elementary stream data unit
	ICC_ElementaryStreamInfo ( see page 30)	
	ICC_StreamRecognizer ( see page 32)	

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
	DTS (see page 28)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 28), except the coded video with B-frames.
	Duration (see page 28)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
	NumSamples (see page 28)	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 28)	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 28)	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 28)	The frame's first sample order number.
	SequenceEntryFlag (see page 29)	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 28), 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

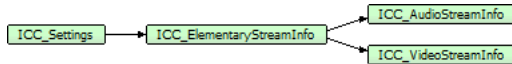
The sequence entry point flag. In the case of mpeg video, it means that SEQUENCE_HEADER presents in the elementary data. This flag is used to signal the multiplexers to generate the entry point (like System Header) at this elementary data.

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;
```

Properties

	Name	Description
	BitRate (see page 30)	The bitrate (max) of the elementary stream.
	FrameRate (see page 30)	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 30)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 112) 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

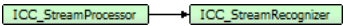
The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE ([see page 112](#)) for details.

Syntax

```
__property CC_ELEMENTARY_STREAM_TYPE * StreamType;
```

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
	GetStreamInfo (see page 32)	

Methods

GetStreamInfo

Syntax

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

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 interface. The interface supports the basic functionality for serialization and the like. The class is the base for all the other specific settings.

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 236) Format field):

```
[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 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 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 fields.

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 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>
```

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.

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` 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>
```


Video API

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

Interfaces

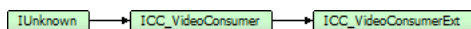
Interfaces

	Name	Description
	ICC_VideoConsumer (see page 44)	Provides the methods specific for the generic video data consumer.
	ICC_VideoProducer (see page 46)	Provides the methods specific for the general video data producers.
	ICC_VideoConsumerExt (see page 49)	Provides the methods specific for the generic video data consumer.
	ICC_VideoDecoder (see page 50)	The default and main interface to control the instance of CC_MpegVideoDecoder class.
	ICC_VideoEncoder (see page 52)	The default and main interface to control the instance of CC_MpegVideoEncoder class.
	ICC_VideoFrameInfo (see page 55)	Interface provides the common description of the video frame.
	ICC_VideoFrameQualityInfo (see page 57)	Provides methods for retrieving the video PSNR (see page 58) (quality measure) information of the video frame.
	ICC_VideoStreamInfo (see page 58)	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 45)	Add another frame to the encoder's input queue.
	GetStride (see page 45)	

	GetVideoFrameInfo (see page 46)	Get the description of the current video frame.
	GetVideoStreamInfo (see page 46)	Get the description of the video stream which is being decoded.
	IsFormatSupported (see page 46)	

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 62](#)) 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 62](#)).

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

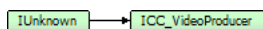
Syntax

```
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 47)	Retrieve the current video frame.
	GetStride (see page 48)	

	GetVideoFrameInfo (see page 48)	Get the description of the current video frame.
	GetVideoStreamInfo (see page 48)	Get the description of the video stream which is being decoded.
	IsFormatSupported (see page 48)	
	IsFrameAvailable (see page 48)	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 62](#)) 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 62](#)).

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  
);
```

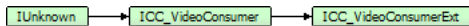
Returns

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 45)	Add another frame to the encoder's input queue.
◆	GetStride (see page 45)	
◆	GetVideoFrameInfo (see page 46)	Get the description of the current video frame.
◆	GetVideoStreamInfo (see page 46)	Get the description of the video stream which is being decoded.
◆	IsFormatSupported (see page 46)	

ICC_VideoConsumerExt Interface

	Name	Description
◆	AddScaleFrame (see page 49)	Add a frame with different size to the processing queue.
◆	IsScaleAvailable (see page 50)	

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

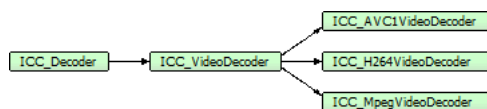
Returns S_OK if successful or an error value otherwise.

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
≡	GetFrame (see page 51)	Retrieve the current video frame.
≡	GetStride (see page 51)	
≡	GetVideoFrameInfo (see page 51)	Get the description of current video frame.
≡	GetVideoStreamInfo (see page 52)	Get the description of video stream which is being decoded.

	IsFormatSupported ( see page 52)	
---	--	--

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 62) 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 62).

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

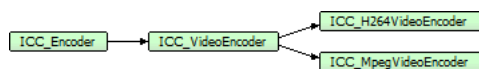
Returns S_OK if data is ready, S_FALSE if not or an error value otherwise.

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
≡	AddFrame (see page 53)	Add another frame to the encoder's input queue.
≡	AddScaleFrame (see page 54)	Add a frame with different size to the processing queue.
≡	GetStride (see page 54)	

	GetVideoFrameInfo (see page 54)	Get the description of current video frame.
	GetVideoStreamInfo (see page 54)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 55)	
	IsScaleAvailable (see page 55)	

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 62](#)) 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 62](#)).

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**

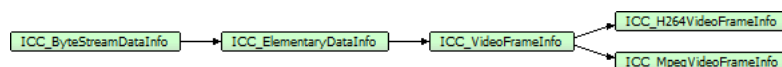
```
HRESULT IsFormatSupported(
    [in] CC_COLOR_FMT fmt,
    [out,retval,defaultvalue(NULL)] CC_BOOL * pResult
);
```

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
	DTS (see page 28)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 28), except the coded video with B-frames.
	Duration (see page 28)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.

	NumSamples (see page 28)	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 28)	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 28)	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 28)	The frame's first sample order number.
	SequenceEntryFlag (see page 29)	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 56)	The number of video frame in the coding order. Zero-based.
	Flags (see page 57)	Various flags of the coded video frame. Value of this field depend of the video stream type.
	FrameType (see page 57)	The frame coding type (see page 64).
	InterlaceType (see page 57)	The field order of video frame.
	Number (see page 57)	The number of video frame in native (display) order. zero-based.
	PictStruct (see page 57)	The picture structure of the MPEG video frame.
	TimeCode (see page 57)	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 depend of the video stream type.

Syntax

```
__property DWORD * Flags;
```

FrameType

The frame coding type ([see page 64](#)).

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.

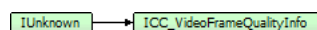
Syntax

```
__property CC_TIMECODE * TimeCode;
```

ICC_VideoFrameQualityInfo Interface

Provides methods for retrieving the video PSNR ([see page 58](#)) (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
	AvgQuantScale (see page 58)	
	PSNR (see page 58)	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;
```

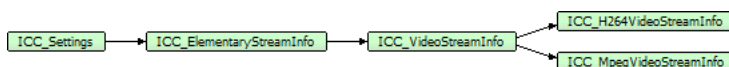
Returns

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;
```

Properties

	Name	Description
	BitRate (see page 30)	The bitrate (max) of the elementary stream.

	FrameRate (see page 30)	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 30)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 112) for details.

ICC_VideoStreamInfo Interface

	Name	Description
	AspectRatio (see page 59)	The aspect ratio cx:cy.
	FrameSize (see page 59)	The size in pixels of video frame.
	ProgressiveSequence (see page 59)	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 61)	The MPEG-2 video chrominance format. (see iso/iec 13818-2 6.3.5 Table 6-5 for details).
	CC_COLOR_FMT (see page 62)	The color format of video frames.
	CC_FRAME_TYPE (see page 64)	Coded video frame type.
	CC_COLOUR_PRIMARIES (see page 65)	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 67)	Symbolic names for CC_GOP_DESCR::N field.
	CC_INTERLACE_TYPE (see page 68)	The field order of video frame.
	CC_MATRIX_COEFFICIENTS (see page 68)	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 69)	The coded macroblock structure.
	CC_PICTURE_STRUCTURE (see page 70)	The coding structure of coded video frame.
	CC_TRANSFER_CHARACTERISTICS (see page 70)	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 72)	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 73)	

Structures

	Name	Description
	CC_COLOUR_DESCRIPTION (see page 65)	The aggregate representation of CC_COLOUR_PRIMARIES (see page 65), CC_TRANSFER_CHARACTERISTICS (see page 70) and CC_MATRIX_COEFFICIENTS (see page 68).
	CC_GOP_DESCR (see page 67)	Group of Pictures (GOP) description.
	CC_QUANT_DESCR (see page 73)	Quantization parameter for defining mquant parameter in VBR modes.
	CC_ADD_VIDEO_FRAME_PARAMS (see page 73)	

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 = 10
};
```

Members

CC_CHROMA_FORMAT_UNKNOWN

Unknown chroma format.

CC_CHROMA_400

Luma-only mode

CC_CHROMA_420

Half vertical, half horizontal color resolution.

CC_CHROMA_422

Full vertical, half horizontal color resolution.

CC_CHROMA_444

Full color resolution.

CC_CHROMA_RGB

RGB, full color resolution

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_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_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_16BIT = 0x00000840,
    CCF_YUV420_16BIT = 0x00000841,
    CCF_YUV422_16BIT = 0x00000842,
    CCF_YUV444_16BIT = 0x00000843,
    CCF_UYVY_10BIT = 0x00000231,
    CCF_YUY2_10BIT = 0x00000232,
    CCF_UYVY_16BIT = 0x00000831,
    CCF_YUY2_16BIT = 0x00000832,
    CCF_YV12 = CCF_YUV420,
    CCF_YV16 = CCF_YUV422,
    CCF_NV12 = 0x00000112,
    CCF_V210 = 0x00000210,
    CCF_SC10 = 0x00000211,
    CCF_Y8 = 0x00000008
};
```

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_RGB64

The color data in the 16-bit BGRA format, 8 bytes per pixel.

CCF_RGB48

The color data in the 16-bit BGR format, 6 bytes per pixel.

CCF_BGR64

The color data in the 16-bit RGBA format, 8 bytes per pixel.

CCF_BGR48

The color data in the 16-bit RGB format, 6 bytes per pixel.

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_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_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_NV12

4:2:0 Planar Y, UV interleaved

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.

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 from 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 65), `CC_TRANSFER_CHARACTERISTICS` (see page 70) and `CC_MATRIX_COEFFICIENTS` (see page 68).

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 65) description.

TC

see `CC_TRANSFER_CHARACTERISTICS` (see page 70) description.

MC

see `CC_MATRIX_COEFFICIENTS` (see page 68) 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  
};
```

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_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

"lbbbbPbbbbP..." - 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  
};
```

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_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

```
[vl_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 62) 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 76)	Provides the methods specific for the generic audio data consumer.
	ICC_AudioProducer (see page 78)	Provides the methods specific for the generic waveform-audio producer.
	ICC_AudioDecoder (see page 81)	The default and main interface to control the instance of CinecoderMpegAudioDecoder class.
	ICC_AudioEncoder (see page 84)	The default and main interface to control the instance of CinecoderMpegAudioEncoder class.
	ICC_AudioFrameInfo (see page 86)	Represents the generic audio stream description.
	ICC_AudioStreamInfo (see page 87)	Represents the generic audio stream description.
	ICC_AudioEncoderSettings (see page 89)	The common settings for any audio encoder initialization.

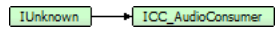
Enumerations

	Name	Description
	CC_AUDIO_FMT (see page 91)	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 76)	Get the description of audio stream which is being decoded.
	ProcessAudio (see page 76)	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.

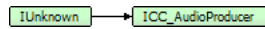
Returns

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 78)	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 79)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 79)	Get the description of the audio stream.
	GetSampleBytes (see page 79)	
	IsFormatSupported (see page 80)	

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

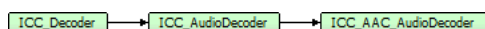
Syntax

```
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
	GetAudio (see page 81)	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 82)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 82)	Get the description of the audio stream.
	GetSampleBytes (see page 83)	
	IsFormatSupported (see page 83)	

Properties

	Name	Description
	NumSamples (see page 83)	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.

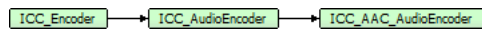
Syntax

```
__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
	GetAudioFrameInfo (see page 84)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 84)	Get the description of audio stream which is being decoded.
	ProcessAudio (see page 85)	Puts another audio uncompressed data to the audio consumer.

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;
```

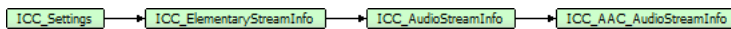
Properties

	Name	Description
	DTS (see page 28)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 28), except the coded video with B-frames.
	Duration (see page 28)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
	NumSamples (see page 28)	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 28)	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 28)	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 28)	The frame's first sample order number.
	SequenceEntryFlag (see page 29)	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;
```

Properties

	Name	Description
	BitRate (see page 30)	The bitrate (max) of the elementary stream.
	FrameRate (see page 30)	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 30)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 112) for details.

ICC_AudioStreamInfo Interface

	Name	Description
	BitsPerSample (see page 87)	The bit depth of one audio sample (of one channel).
	ChannelMask (see page 88)	The mask of channels. See the CC_AUDIO_CHANNEL_MASK for details
	NumChannels (see page 88)	The number of channels in the waveform-audio data.
	SampleRate (see page 88)	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.

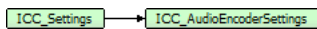
Syntax

```
__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;
```

Properties

	Name	Description
	BitRate (see page 89)	The bitrate of entire audio bitstream or corresponding audio frame.
	BitsPerSample (see page 89)	The bit depth of one audio sample (of one channel).
	FrameRate (see page 90)	The number of audio frames per second.
	NumChannels (see page 90)	The number of channels in the waveform-audio data.
	SampleRate (see page 90)	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 95)	Interface provides the methods specific for general stream demultiplexer object.
	ICC_Multiplexer (see page 98)	Interface provides the methods specific for general stream multiplexer object.
	ICC_BaseMultiplexerPinSettings (see page 101)	
	ICC_DemultiplexedDataCallback (see page 103)	Unified demultiplexer data output - for Program & Transport demultiplexers.
	ICC_MultiplexedStreamInfo (see page 105)	
	ICC_MultiplexedDataDescr (see page 107)	The description of data coming from the multiplexer (via ICC_StreamDescr::DataInfo property).
	ICC_BaseMultiplexerSettings (see page 109)	

Enumerations

	Name	Description
	CC_ELEMENTARY_STREAM_TYPE (see page 112)	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 116)	The stream types.
	CC_MUX_OUTPUT_POLICY (see page 117)	
	CC_PES_ID (see page 119)	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 120)	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 121)	
---	--	--

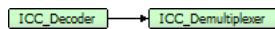
Structures

	Name	Description
	CC_ACCESS_UNIT_DESCR (see page 111)	Description of the multiplexed elementary stream data unit.
	CC_PACKET_DESCR (see page 118)	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
	CatchStream (see page 95)	
	GetData (see page 96)	The method to obtain the demultiplexed packet directly (not via callback).
	GetStreamInfo (see page 96)	
	ReleaseAllStreams (see page 96)	
	ReleaseStream (see page 97)	

Properties

	Name	Description
	DataSize (see page 97)	
	SCR (see page 97)	The last clock reference (SCR or PCR) read from the multiplexed stream
	StreamID (see page 97)	

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

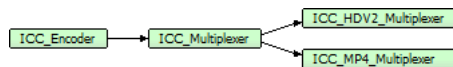
Syntax

```
__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
	CreatePin (see page 98)	Method creates an input pin to add the specified type of elementary data needed to be multiplexed.
	CreatePinByXml (see page 99)	Method creates an input pin to add the specified type of elementary data needed to be multiplexed.
	GetPin (see page 99)	

Properties

	Name	Description
	PinCount (see page 100)	/// Method creates an input pin to add the specified type of elementary data needed to be multiplexed. /// Returns: Returns S_OK if successful or error code otherwise. HRESULT CreatePinByType([in] CC_ELEMENTARY_STREAM_TYPE (see page 112) stream_type, // Pin type [out,retval] ICC_ByteStreamConsumer **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 multiplexed.

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 multiplexed.

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

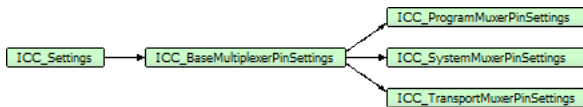
/// Method creates an input pin to add the specified type of elementary data needed to be multiplexed. /// Returns: Returns S_OK if successful or error code otherwise. HRESULT CreatePinByType([in] CC_ELEMENTARY_STREAM_TYPE (see page 112) stream_type, // Pin type [out,retval] ICC_ByteStreamConsumer **pOutput // 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;
```

Properties

	Name	Description
	BasePTS (see page 101)	
	BitRate (see page 101)	
	DataAlignPeriod (see page 102)	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 102)	
	SampleRate (see page 102)	
	StreamID (see page 102)	
	StreamType (see page 102)	

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

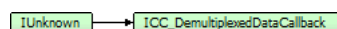
Syntax

```
__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 103)	

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

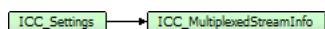
The presentation time stamp of the first unit commencing in the packet. If no units commencing, the value of CC_NO_TIME will be assigned.

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
	GetProgram (see page 105)	

Properties

	Name	Description
 R	BitRate (see page 105)	
 R	NumPrograms (see page 106)	
 R	StreamID (see page 106)	
 R	StreamType (see page 106)	

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

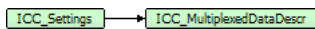
Syntax

```
__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
	GetAccessUnitInfo (? see page 107)	
	GetPacketInfo (? see page 107)	

Properties

	Name	Description
	NumAccessUnits (? see page 108)	
	NumPackets (? see page 108)	

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;
```

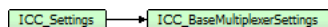
NumPackets

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;
```

Properties

	Name	Description
	AsyncInputMode (see page 109)	
	BitRate (see page 109)	Target bitrate of the multiplexed stream. If zero - calculates automatically.
	InitialPTS (see page 110)	Clock reference for the first byte of multiplexed stream.
	MaxOutputBlkSize (see page 110)	
	OutputPolicy (see page 110)	The output policy for multiplexer.
	RateMode (see page 110)	Target bitrate mode - constant or variable.
	StreamType (see page 110)	Target multiplexed stream type.
	TrailingAlignment (see page 110)	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.

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 = 0xD0,
    CC_ES_TYPE_VIDEO_DVCPRO = 0xD1,
    CC_ES_TYPE_VIDEO_DNxHD = 0xD2,
    CC_ES_TYPE_VIDEO_AVC_INTRA = 0x11b,
    CC_ES_TYPE_VIDEO_PRORES = 0x120,
    CC_ES_TYPE_EBU_TELETEXT = 0x4206,
    CC_ES_TYPE_SMPTE_436 = 0x436,
    CC_ES_TYPE_VIDEO_DANIEL = 0xD206,
    CC_ES_TYPE_VIDEO_Y4M = 0xF406
};
```

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_SMPTE_436

SMPTE 436 Encoded VANC data

CC_ES_TYPE_VIDEO_DANIEL

Daniel Video Stream.

CC_ES_TYPE_VIDEO_Y4M

Y4M Video Stream

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 126)	
	CC_MPG_INTRA_VLC_TABLE (see page 127)	The intra_vlc_format value. Refer ISO/IEC 13818-2 7.2.2.1 for details.
	CC_MPG_MB_SCAN_PATTERN (see page 127)	The block's scan patterns.
	CC_MPG_PROFILE_LEVEL (see page 128)	
	CC_MPG_QUANT_SCALE_TYPE (see page 129)	
	CC_MPG_ASPECT_RATIO_CODE (see page 130)	Frame aspect ratios.
	CC_MPG_MOTION_PARAMS (see page 130)	

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

```
[vl_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

```
[vl_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_HALFPPEL = 0x00000000,
CC_MPG_ME_NO_HALFPPEL = 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_HALFPPEL,
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_HALFPPEL|CC_MPG_ME_ALIGN_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_HALFPPEL|CC_MPG_ME_ALIGN_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 134)	Provides the particular MPEG video frame description.
	ICC_MpegVideoStreamInfo (see page 137)	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 136)	Retrieves the specified user data which belongs to the video frame.

Properties






	Name	Description
	DTS (see page 28)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 28), except the coded video with B-frames.
	Duration (see page 28)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
	NumSamples (see page 28)	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 28)	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 28)	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 28)	The frame's first sample order number.
	SequenceEntryFlag (see page 29)	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 56)	The number of video frame in the coding order. Zero-based.
	Flags (see page 57)	Various flags of the coded video frame. Value of this field depend of the video stream type.
	FrameType (see page 57)	The frame coding type (see page 64).
	InterlaceType (see page 57)	The field order of video frame.
	Number (see page 57)	The number of video frame in native (display) order. zero-based.
	PictStruct (see page 57)	The picture structure of the MPEG video frame.
	TimeCode (see page 57)	The timecode of video frame.

ICC_MpegVideoFrameInfo Interface

	Name	Description
	IntraDcPrec (see page 136)	The intra_dc_precision (8-11 bits)
	SecondFieldInfo (see page 136)	Retrieves the second field's information of a frame, if it exist (i.e. frame encoded in a field mode)
	TempRef (see page 137)	The frame's temporal reference.
	UserDataCount (see page 137)	The number of MPEG_USER_DATA, associated with the video frame.
	VBV_Delay (see page 137)	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.

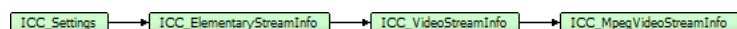
Syntax

```
__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

ICC_MpegVideoStreamInfo Interface




	Name	Description
	GetUserData (see page 139)	Retrieves the specified user data which associated with the stream (seq_hdr level).

Properties













	Name	Description
	BitRate (see page 30)	The bitrate (max) of the elementary stream.
	FrameRate (see page 30)	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 30)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 112) for details.
---	--	---

ICC_VideoStreamInfo Interface

	Name	Description
	AspectRatio (see page 59)	The aspect ratio cx:cy.
	FrameSize (see page 59)	The size in pixels of video frame.
	ProgressiveSequence (see page 59)	If TRUE - all frames in the stream coded without fields (progressive).

ICC_MpegVideoStreamInfo Interface

	Name	Description
	AspectRatioCode (see page 139)	The Aspect Ratio code.
	BitRate (see page 139)	The bitrate of video bitstream.
	ChromaFormat (see page 139)	Chroma format.
	ColorCoefs (see page 140)	The color transformation description.
	DisplaySize (see page 140)	The Display Size (resulting frame size).
	FrameDuration (see page 140)	The duration of one frame, in CC_TIMEBASE units.
	Layer (see page 140)	The layer of MPEG video (1 or 2).
	Mpeg1ConstrainedFlag (see page 140)	MPEG-1 constrained stream flag.
	ProfileAndLevel (see page 140)	Profile and level.
	UserDataCount (see page 140)	The number of MPEG_USER_DATA, associated with the video frame.
	VBV_BufferSize (see page 140)	The VBV buffer size.
	VideoFormat (see page 141)	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 transformation 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

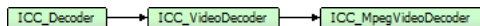
Interfaces

	Name	Description
	ICC_MpegVideoDecoder (see page 142)	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
	GetFrame (see page 51)	Retrieve the current video frame.
	GetStride (see page 51)	
	GetVideoFrameInfo (see page 51)	Get the description of current video frame.
	GetVideoStreamInfo (see page 52)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 52)	

ICC_MpegVideoDecoder Interface

	Name	Description
	GetMpegVideoFrameInfo (see page 143)	Get the MPEG-specific description of current video frame.
	GetMpegVideoStreamInfo (see page 143)	Get the MPEG-specific description of video stream which is being decoded.
	SetElementaryDataCallback (see page 144)	

Properties

ICC_MpegVideoDecoder Interface

	Name	Description
	PictureDecodingLevel ( see page 144)	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 144)	The affinity mask for object's threads. 0 = default (current process) affinity mask.
	ThreadsCount ( see page 144)	Maximal number of threads which object can use. 0 = automatic.
	ThreadsPriority ( see page 145)	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 priority for object's threads.

Syntax

```
__property CC_PRIORITY ThreadsPriority;
```

MPEG Video Encoder

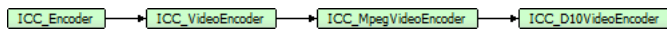
Interfaces

	Name	Description
	ICC_MpegVideoEncoder (see page 146)	The default and main interface to control the instance of CinecoderMpegVideoEncoder class.
	ICC_MpegVideoEncoderSettings (see page 149)	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
	AddFrame (see page 53)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 54)	Add a frame with different size to the processing queue.
	GetStride (see page 54)	
	GetVideoFrameInfo (see page 54)	Get the description of current video frame.
	GetVideoStreamInfo (see page 54)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 55)	
	IsScaleAvailable (see page 55)	





ICC_MpegVideoEncoder Interface

	Name	Description
	AddUserData (see page 147)	Adds user data for the subsequent video frame.

	GetMpegVideoFrameInfo (see page 148)	Get the MPEG-specific description of current video frame.
---	---	---

Properties

ICC_MpegVideoEncoder Interface

	Name	Description
	InitialTimeCode (see page 148)	Specifies the initial timecode for the first frame of the first GOP. Should be used before first call to AddFrame (see page 53).
	ThreadsAffinity (see page 148)	The affinity mask for object's threads. 0 = default (current process) affinity mask.
	ThreadsCount (see page 148)	Maximal number of threads which object can use. 0 = automatic.
	ThreadsPriority (see page 148)	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 53).

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

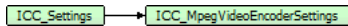
The priority for object's threads.

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
	AddUserData (see page 152)	Adds another user data to the Sequence Layer (see page 156) of the MPEG video stream.
	GetQuantMatrix (see page 152)	
	GetUserData (see page 152)	Retrieves the specified user data which associated with the stream (seq_hdr level).
	SetQuantMatrix (see page 153)	

Properties

	Name	Description
	AspectRatioCode (see page 153)	See CC_MPG_ASPECT_RATIO_CODE (see page 130).
	AvgBitRate (see page 153)	The average bitrate in VBR mode.
	BitRate (see page 153)	The target bitrate in CBR mode or max bitrate in VBR mode.
	BlocksPerSlice (see page 153)	The max number of macroblocks per slice.
	BlurFilterCoef (see page 153)	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 154)	The chroma resolution format (4:2:0, 4:2:2 or 4:4:4).
	ClosedGOPs (see page 154)	Make all of GOPs "closed" - without backward referencing at the beginning of GOP (see page 155).

	ColorCoefs (see page 154)	
	DisableSceneDetector (see page 154)	Enable/disable the built-in scene change detector (which is ON by default).
	DisplaySize (see page 154)	The DisplaySize field of seq_disp_extension.
	EncodingLatency (see page 154)	Controls the latency of encoder (in frames) (0 = auto, 1 = min).
	FixedGopStructure (see page 154)	Fix the structure of a GOP (see page 155), even for the first GOP (see page 155) in stream.
	FrameRate (see page 155)	The frame's rate of video stream.
	FrameSize (see page 155)	The physical frame size, in pixels.
	GOP (see page 155)	The GOP settings. See CC_GOP_DESCR (see page 67) for details.
	InitialTimeCode (see page 155)	Specifies the initial timecode for the first frame of the first GOP (see page 155).
	InterlaceType (see page 155)	The video field order.
	IntraDCPrecision (see page 155)	The precision of intra DC coefficient (8-11 bits).
	IntraRefresh (see page 155)	If true,
	IntraVLCTable (see page 155)	A VLC table for encoding intra DC coefficients.
	Layer (see page 156)	The layer of the target MPEG coded video stream. Default is 2 (MPEG-2).
	LowDelay (see page 156)	Low delay mode
	MB_ScanPattern (see page 156)	The scan pattern (zig-zag or alternate).
	MB_Struct (see page 156)	Field/frame macrobloc structure.
	MEQuality (see page 156)	The motion estimation quality. Value in range 0..100.
	MinBitRate (see page 156)	The minimal bitrate in VBR mode (default is 0).
	MotionParams (see page 156)	See CC_MPG_MOTION_PARAMS (see page 130) for details.
	PictureStructure (see page 157)	The picture structure.

	ProfileAndLevel ( see page 157)	The profile@level of the destination stream. Restricted by the license.
	ProgressiveSequence ( see page 157)	If true, all frames in the stream coded without fields (progressive).
	QuantFunc ( see page 157)	Quantization function
	QuantMatrixPictureLevel ( see page 157)	Allows writing the quant matrix extension header at every specified picture type. CC_FRAME_TYPE_UNKNOWN disables it.
	QuantScale ( see page 157)	The average quantization scale in VBR mode.
	QuantScaleType ( see page 157)	Linear or non-linear VLC coefficients scale type.
	RateMode ( see page 158)	The mode of bitrate controller - see CC_BITRATE_MODE.
	SequenceHeaderPeriod ( see page 158)	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 155). You can also specify the period in milliseconds by adding the FRQ_TIMEVAL_MS flag to the value.
	SuppressSeqEndCode ( see page 158)	Put sequence_end_code after the last coded picture.
 	UserDataCount ( see page 158)	The number of MPEG_USER_DATA, associated with the stream.
	VBV_BufferSize ( see page 158)	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 158)	The video format.

Methods

AddUserData

Adds another user data to the Sequence Layer ( see page 156) 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 130).

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

155).

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 155), even for the first GOP (see page 155) 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 67) for details.

Syntax

```
__property CC_GOP_DESCR GOP;
```

InitialTimeCode

Specifies the initial timecode for the first frame of the first GOP (see page 155).

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 130) 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 155). 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

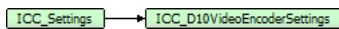
Interfaces

	Name	Description
	ICC_D10VideoEncoderSettings (see page 159)	The settings for CinecoderD10VideoEncoder initialization.
	ICC_D10VideoEncoder (see page 162)	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
	AddUserData (see page 160)	Adds another user data to the Sequence Layer of a MPEG video stream.
	GetUserData (see page 160)	Retrieves the specified user data associated with the stream (seq_hdr level).

Properties

	Name	Description
	AspectRatioCode (see page 161)	Optional. See CC_ASPECT_RATIO.
	BitRate (see page 161)	Required. The bitrate of target IMX stream. Possible values are 30000000, 40000000 and 50000000 bits/sec.
	ColorCoefs (see page 161)	The color transformation description.
	FrameSize (see page 161)	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 161)	Quantization function
 R	UserDataCount ( see page 162)	The number of MPEG_USER_DATA, associated with the stream.
	VideoFormat ( see page 162)	The IMX video format (CC_VIDEO_FORMAT_PAL or CC_VIDEO_FORMAT_NTSC). If no FrameSize ( see page 161) 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 161) is specified, 720x608 or 720x512 are assumed.

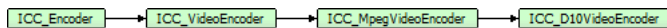
Syntax

```
__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
≡	AddFrame (see page 53)	Add another frame to the encoder's input queue.
≡	AddScaleFrame (see page 54)	Add a frame with different size to the processing queue.
≡	GetStride (see page 54)	
≡	GetVideoFrameInfo (see page 54)	Get the description of current video frame.
≡	GetVideoStreamInfo (see page 54)	Get the description of video stream which is being decoded.
≡	IsFormatSupported (see page 55)	
≡	IsScaleAvailable (see page 55)	

ICC_MpegVideoEncoder Interface

	Name	Description
	AddUserData ( see page 147)	Adds user data for the subsequent video frame.
	GetMpegVideoFrameInfo ( see page 148)	Get the MPEG-specific description of current video frame.

Properties**ICC_MpegVideoEncoder Interface**

	Name	Description
	InitialTimeCode ( see page 148)	Specifies the initial timecode for the first frame of the first GOP. Should be used before first call to AddFrame ( see page 53).
	ThreadsAffinity ( see page 148)	The affinity mask for object's threads. 0 = default (current process) affinity mask.
	ThreadsCount ( see page 148)	Maximal number of threads which object can use. 0 = automatic.
	ThreadsPriority ( see page 148)	The priority for object's threads.












MPEG Audio

MPEG Audio Encoder

MPEG Audio Decoder

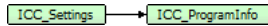
MPEG Multiplex

Interfaces

	Name	Description
	ICC_ProgramInfo ( see page 168)	
	ICC_SystemDescriptorsManager ( see page 170)	The extension to ICC_SystemDescriptorsReader ( see page 178), which allows you to add and remove the descriptors.
	ICC_PES_Info ( see page 173)	
	ICC_TS_ProgramDescr ( see page 176)	
	ICC_SystemDescriptorsReader ( see page 178)	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
	GetStream (see page 168)	

Properties

	Name	Description
 R	Descriptors (see page 169)	
 R	NumStreams (see page 169)	
 R	PCR_PID (see page 169)	
 R	PMT_PID (see page 169)	
 R	ProgNum (see page 169)	

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

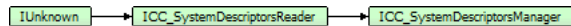
Syntax

```
__property WORD* ProgNum;
```

ICC_SystemDescriptorsManager Interface

The extension to ICC_SystemDescriptorsReader (see page 178), 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 178)	Retrieves the specified descriptor by its ordinal number.
	IndexOf (see page 179)	Retrieves the descriptor index with the specified code.
	StoreToBuffer (see page 179)	Get (see page 178) all stored descriptors as raw data.

ICC_SystemDescriptorsManager Interface

	Name	Description
	Add (see page 171)	Add the new descriptor to the object.
	AddFromBuffer (see page 171)	Add (see page 171) descriptors from raw buffer. Descriptors must be collocated, in form [code:byte][length:byte][data:byte[]]...
	Clear (see page 172)	Empties the list of descriptors.
	CopyFrom (see page 172)	Add (see page 171) all of descriptors from another object.
	Remove (see page 172)	The method removes the specified descriptor from a list.

Properties

	Name	Description
	NumDescr (see page 180)	Retrieves the number of descriptors in the list.
	Size (see page 180)	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 171) 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 171) 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

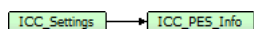
The descriptor index to be removed.

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;
```

Properties

	Name	Description
	AdditionalCopyInfo (see page 174)	
	BasePTS (see page 174)	
	CrcProtected (see page 174)	
	Descriptors (see page 174)	
	OriginalFlag (see page 174)	
	PacketHeaderSize (see page 174)	
	PacketSize (see page 174)	
	PID (see page 174)	
	PriorityFlag (see page 174)	
	ScramblingControl (see page 175)	
	StreamID (see page 175)	
	StreamType (see page 175)	

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;
```

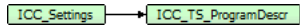
StreamType

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;
```

Properties

	Name	Description
	CrcProtected (see page 176)	
	Descriptors (see page 177)	
	DiscontinuityThreshold (see page 177)	
	InitialPTS (see page 177)	
	PCR_Period (see page 177)	
	PCR_PID (see page 177)	
	PMT_Period (see page 177)	
	PMT_PID (see page 177)	
	ProgNum (see page 177)	

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

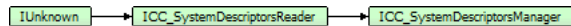
Syntax

```
__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 178)	Retrieves the specified descriptor by its ordinal number.
	IndexOf (see page 179)	Retrieves the descriptor index with the specified code.
	StoreToBuffer (see page 179)	Get (see page 178) all stored descriptors as raw data.

Properties

	Name	Description
	NumDescr (see page 180)	Retrieves the number of descriptors in the list.
	Size (see page 180)	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_SYSDSCR * 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 will be found, -1 will be returned.

Returns

S_OK if found and S_FALSE if not found.

StoreToBuffer

Get (see page 178) 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' raw 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

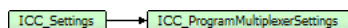
	Name	Description
	ICC_ProgramMultiplexerSettings (see page 181)	
	ICC_ProgramMuxerPinSettings (see page 184)	

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
	AddStream (see page 182)	
	GetStream (see page 182)	

Properties

	Name	Description
	BitRate (see page 183)	
	CrcProtected (see page 183)	
	Descriptors (see page 183)	

	NumStreams (see page 183)	
	PacketAlignment (see page 183)	
	PacketHeaderSize (see page 183)	
	PacketSize (see page 183)	
	PackHeaderPeriod (see page 183)	
	PayloadAlignment (see page 183)	
	PutProgramStreamMap (see page 184)	
	RateMode (see page 184)	
	StandaloneSystemHeader (see page 184)	
	SuppressIsoEndCode (see page 184)	
	SystemHeaderPeriod (see page 184)	
	TrailingAlignment (see page 184)	

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;
```

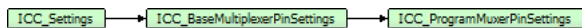
TrailingAlignment

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;
```

Properties

	Name	Description
	BasePTS (see page 101)	
	BitRate (see page 101)	
	DataAlignPeriod (see page 102)	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 102)	
	SampleRate (see page 102)	

	StreamID (see page 102)	
	StreamType (see page 102)	

ICC_ProgramMuxerPinSettings Interface

	Name	Description
	AdditionalCopyInfo (see page 185)	
	CrcProtected (see page 186)	
 R	Descriptors (see page 186)	
	HideFromSystemHeader (see page 186)	
	OriginalFlag (see page 186)	
	PacketAlignment (see page 186)	
	PacketHeaderSize (see page 186)	
	PacketSize (see page 186)	
	PayloadAlignment (see page 186)	
	PriorityFlag (see page 186)	
	ScramblingControl (see page 186)	

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

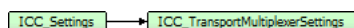
	Name	Description
	ICC_TransportMultiplexerSettings (see page 187)	
	ICC_TransportMuxerPinSettings (see page 189)	
	ICC_M2TSMP_MultiplexerSettings (see page 191)	

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;
```

Properties

	Name	Description
	BitRate (see page 188)	
	CrcProtected (see page 188)	
	Descriptors (see page 188)	
	PAT_Period (see page 188)	
	PCR_Period (see page 188)	
	PCR_PID (see page 188)	

	PMT_Period (see page 189)	
	PMT_PID (see page 189)	
	ProgNum (see page 189)	
	RateMode (see page 189)	
	StreamID (see page 189)	
	TrailingAlignment (see page 189)	

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;
```

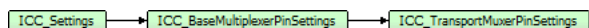
TrailingAlignment

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;
```

Properties

	Name	Description
	BasePTS (see page 101)	
	BitRate (see page 101)	
	DataAlignPeriod (see page 102)	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 102)	
	SampleRate (see page 102)	

	StreamID (see page 102)	
	StreamType (see page 102)	

ICC_TransportMuxerPinSettings Interface

	Name	Description
	AdditionalCopyInfo (see page 190)	
	CrcProtected (see page 190)	
 R	Descriptors (see page 191)	
	OriginalFlag (see page 191)	
	PesHeaderPeriod (see page 191)	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 191)	
	PriorityFlag (see page 191)	
	ScramblingControl (see page 191)	

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;
```

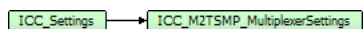
ScramblingControl

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
	AddProgram ( see page 192)	
	GetProgram ( see page 192)	

Properties

	Name	Description
	BitRate ( see page 193)	

	CrcProtected (see page 193)	
	NumPrograms (see page 193)	
	PAT_Period (see page 193)	
	PCR_Period (see page 193)	
	PMT_Period (see page 193)	
	ProgNumBase (see page 193)	
	RateMode (see page 193)	
	StreamID (see page 193)	
	TrailingAlignment (see page 193)	

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;
```

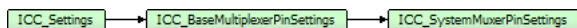
MPEG-1 System Stream

Interfaces

	Name	Description
	ICC_SystemMuxerPinSettings (see page 194)	
	ICC_SystemMuxerSettings (see page 195)	

ICC_SystemMuxerPinSettings Interface








Class Hierarchy




Syntax

```
[object, uuid(00001B11-be08-11dc-aa88-005056c00008), pointer_default(unique), local]  
interface ICC_SystemMuxerPinSettings : ICC_BaseMultiplexerPinSettings;
```

Properties

	Name	Description
	BasePTS (see page 101)	
	BitRate (see page 101)	
	DataAlignPeriod (see page 102)	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 102)	
	SampleRate (see page 102)	
	StreamID (see page 102)	
	StreamType (see page 102)	

ICC_SystemMuxerPinSettings Interface

	Name	Description
	PacketAlignment (see page 195)	

	PacketHeaderSize (see page 195)	
	PacketSize (see page 195)	
	PayloadAlignment (see page 195)	

Properties

PacketAlignment

Syntax

```
__property [in] PacketAlignment;
```

PacketHeaderSize

Syntax

```
__property [in] PacketHeaderSize;
```

PacketSize

Syntax

```
__property [in] PacketSize;
```

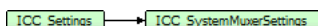
PayloadAlignment

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;
```

Properties

	Name	Description
	BitRate (see page 196)	
	PacketAlignment (see page 196)	
	PacketHeaderSize (see page 196)	

	PacketSize (see page 196)	
	PackHeaderPeriod (see page 196)	
	PayloadAlignment (see page 197)	
	RateMode (see page 197)	
	StandaloneSystemHeader (see page 197)	
	SuppressIsoEndCode (see page 197)	
	SystemHeaderPeriod (see page 197)	
	TrailingAlignment (see page 197)	

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;
```

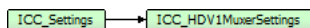
HDV-1 Multiplexer

Interfaces

	Name	Description
	ICC_HDV1MuxerSettings (see page 198)	

ICC_HDV1MuxerSettings Interface

Class Hierarchy



Syntax

```
[object, uuid(00001D01-be08-11dc-aa88-005056c00008), pointer_default(unique), local]  
interface ICC_HDV1MuxerSettings : ICC_Settings;
```

Properties

	Name	Description
	ProgNum (see page 198)	
	StreamID (see page 198)	

Properties

ProgNum

Syntax

```
property [in] ProgNum;
```



StreamID

Syntax

```
property [in] StreamID;
```

HDV-2 Multiplexer

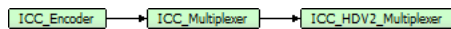
Interfaces

	Name	Description
	ICC_HDV2_Multiplexer (see page 199)	This interface provides a methods specific for the HDV2_Multilexer class.
	ICC_HDV2MuxerSettings (see page 200)	

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
	CreatePin (see page 98)	Method creates an input pin to add the specified type of elementary data needed to be multiplexed.
	CreatePinByXml (see page 99)	Method creates an input pin to add the specified type of elementary data needed to be multiplexed.
	GetPin (see page 99)	

Properties

	Name	Description
	PinCount (see page 100)	/// Method creates an input pin to add the specified type of elementary data needed to be multiplexed. /// Returns: Returns S_OK if successful or error code otherwise. HRESULT CreatePinByType([in] CC_ELEMENTARY_STREAM_TYPE (see page 112) stream_type, // Pin type [out,retval] ICC_ByteStreamConsumer **pOutput // Address where pointer to the created object will be stored.);

ICC_HDV2_Multiplexer Interface

	Name	Description
	InitialTimeCode (? see page 200)	

Properties

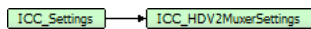
InitialTimeCode

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;
```

Properties

	Name	Description
	ProgNum (? see page 200)	
	StreamID (? see page 200)	

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 ([link](#) see page 201).

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 207)	H264 deblocking filter modes
	CC_H264_DIRECT_PRED_MODE (see page 207)	
	CC_H264_ENTROPY_CODING_MODE (see page 207)	
	CC_H264_FRAME_FLAGS (see page 208)	
	CC_H264_MOTION_FUNC (see page 208)	
	CC_H264_PROFILE (see page 208)	
	CC_H264_SCALING_MATRIX (see page 209)	Scaling matrices for 8x8 quantization.
	CC_H264_SUBBLOCK_SPLIT_MODE (see page 209)	

Structures

	Name	Description
	CC_H264_DEBLOCKING_FILTER_DESCR (see page 206)	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 210)	Provides the particular H.264 video frame description.
	ICC_H264VideoStreamInfo (see page 212)	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 211)	Retrieves the specified user data which belongs to the video frame.

Properties


	Name	Description
	DTS (see page 28)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 28), except the coded video with B-frames.
	Duration (see page 28)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
	NumSamples (see page 28)	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 28)	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 28)	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 28)	The frame's first sample order number.
	SequenceEntryFlag (see page 29)	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 56)	The number of video frame in the coding order. Zero-based.
	Flags (see page 57)	Various flags of the coded video frame. Value of this field depend of the video stream type.
	FrameType (see page 57)	The frame coding type (see page 64).
	InterlaceType (see page 57)	The field order of video frame.
	Number (see page 57)	The number of video frame in native (display) order. zero-based.
	PictStruct (see page 57)	The picture structure of the MPEG video frame.
	TimeCode (see page 57)	The timecode of video frame.

ICC_H264VideoFrameInfo Interface

	Name	Description
	UserDataCount (see page 212)	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.

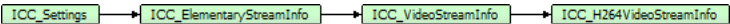
Syntax

```
__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;
```

Properties









	Name	Description
	BitRate (see page 30)	The bitrate (max) of the elementary stream.

	FrameRate (see page 30)	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 30)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 112) for details.

ICC_VideoStreamInfo Interface

	Name	Description
	AspectRatio (see page 59)	The aspect ratio cx:cy.
	FrameSize (see page 59)	The size in pixels of video frame.
	ProgressiveSequence (see page 59)	If TRUE - all frames in the stream coded without fields (progressive).

ICC_H264VideoStreamInfo Interface

	Name	Description
	BitDepthChroma (see page 214)	The chroma samples' bit depth.
	BitDepthLuma (see page 214)	The luma samples' bit depth
	BitRate (see page 214)	The video bitstream bitrate.
	ChromaFormat (see page 214)	Chroma format.
	ColorCoefs (see page 214)	The color transformation description.
	Level (see page 214)	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 214)	The Profile of an H.264 stream. For more information please refer to the H.264 standard Annex A.
	VideoFormat (see page 215)	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 transformation 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

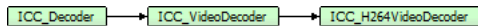
Interfaces

	Name	Description
	ICC_H264VideoDecoder (see page 216)	The interface gives the control to the instance of CC_H264VideoDecoder class.
	ICC_AVC1VideoDecoder (see page 216)	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

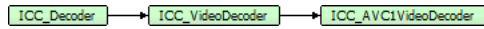
	Name	Description
	GetFrame (see page 51)	Retrieve the current video frame.
	GetStride (see page 51)	
	GetVideoFrameInfo (see page 51)	Get the description of current video frame.
	GetVideoStreamInfo (see page 52)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 52)	

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
⇒	GetFrame (see page 51)	Retrieve the current video frame.
⇒	GetStride (see page 51)	
⇒	GetVideoFrameInfo (see page 51)	Get the description of current video frame.
⇒	GetVideoStreamInfo (see page 52)	Get the description of video stream which is being decoded.
⇒	IsFormatSupported (see page 52)	

H.264 Video Encoder

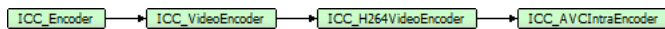
Interfaces

	Name	Description
	ICC_H264VideoEncoder (see page 218)	Interface gives the control to the instance of CC_H264VideoEncoder class.
	ICC_H264VideoEncoderSettings (see page 220)	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
	AddFrame (see page 53)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 54)	Add a frame with different size to the processing queue.
	GetStride (see page 54)	
	GetVideoFrameInfo (see page 54)	Get the description of current video frame.
	GetVideoStreamInfo (see page 54)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 55)	
	IsScaleAvailable (see page 55)	

ICC_H264VideoEncoder Interface

	Name	Description
	AddUserData (see page 219)	Adds user data for the subsequent video frame.

	GetH264VideoFrameInfo (see page 220)	
---	---	--

Properties**ICC_H264VideoEncoder Interface**

	Name	Description
	ThreadsAffinity (see page 220)	The affinity mask for object's threads. 0 = default (current process) affinity mask
	ThreadsCount (see page 220)	Maximum number of threads which object can use. 0 = automatic.
	ThreadsPriority (see page 220)	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 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 priority for object's threads.

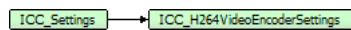
Syntax

```
__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;
```

Properties

	Name	Description
	AspectRatio ( see page 223)	The aspect ratio cx:cy.
	AvgBitRate ( see page 223)	The average bitrate in VBR mode.

	BitDepthChroma (see page 223)	The chroma samples' bit depth (8-12).
	BitDepthLuma (see page 223)	The luma samples' bit depth (8-12).
	BitRate (see page 223)	The target bitrate in CBR mode or max bitrate in VBR mode.
	BlurFilterCoef (see page 223)	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 224)	The chroma resolution format (4:2:0, 4:2:2 or 4:4:4).
	ChromaQPOffset (see page 224)	The offset from the luma QP (see page 227) value for chroma components
	ColorCoefs (see page 224)	Sequence display extension - color coefficients.
	CpbSize (see page 224)	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 224)	Deblocking filter params.
	Deinterlace (see page 224)	Deinterlace method
	DirectPredMode (see page 224)	Direct Prediction Mode.
	DisableSceneDetector (see page 225)	Enable/disable the built-in scene change detector (which is ON by default).
	Enable8x8Transform (see page 225)	
	EntropyCodingMode (see page 225)	The entropy coding method.
	FrameRate (see page 225)	The frame rate of video stream.
	FrameSize (see page 225)	The physical frame size, in pixels.
	GOP (see page 225)	The GOP settings. See CC_GOP_DESCR (see page 67) for details.
	IDR_Period (see page 225)	The Instantaneous Decoding Refresh period.
	InitialCpbLevel (see page 225)	The initial CPB fullness. You can specify it either in bytes or milliseconds.

	InterlaceType ( see page 226)	The field order video.
	Level ( see page 226)	The level of destination stream.
	MB_Struct ( see page 226)	Field/frame macrobloc structure.
	MinBitRate ( see page 226)	The minimal bitrate in VBR mode (default is 0).
	MotionFunc ( see page 226)	The motion estimation method.
	MotionWindow ( see page 226)	The motion estimation window.
	NumRefFrames ( see page 226)	The number of reference frames.
	NumSlices ( see page 227)	The max number of slices per frame.
	PictureStructure ( see page 227)	The picture structure.
	Profile ( see page 227)	The destination stream profile.
	PutAccessUnitDelimiter ( see page 227)	Toggle access unit delimiter (nal_unit_type = 9) on/off.
	PutSeqEndCode ( see page 227)	Put sequence_end_code after the last coded picture.
	QP ( see page 227)	The initial quantization parameter for VBR mode.
	QPPRimeY0TransformBypass ( see page 227)	
	RateMode ( see page 227)	The mode of bitrate controller - see CC_BITRATE_MODE.
	ScalingMatrix ( see page 228)	Standard/Default scaling matrix for 8x8 transform.
	SequenceHeaderPeriod ( see page 228)	You can also specify the period in milliseconds by adding the FRQ_TIMEVAL_MS flag to the value.
	SubBlockSplitMode ( see page 228)	The subblock split mode.
	UseWeightedBiPrediction ( see page 228)	Weighted BiPrediction.
	UseWeightedPrediction ( see page 228)	Weighted Prediction.
	VideoFormat ( see page 228)	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 227) 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 67) 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 231)	Interface for AVC Intra video encoder
	ICC_AVCIntraEncoderSettings ( see page 232)	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 230)	

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,
```

```

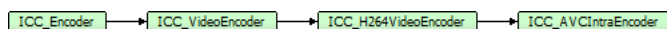
    CC_AVCI_100_1080I_5994 = 0x21
};

```

ICC_AVCIIntraEncoder Interface

Interface for AVC Intra video encoder

Class Hierarchy










Syntax

```



[object, uuid(0242b581-180a-430e-bb9c-1a78e60de3e5), pointer_default(unique), local]
interface ICC_AVCIIntraEncoder : ICC_H264VideoEncoder;

```

Methods


	Name	Description
	AddFrame (see page 53)	Add another frame to the encoder's input queue.
	AddScaleFrame (see page 54)	Add a frame with different size to the processing queue.
	GetStride (see page 54)	
	GetVideoFrameInfo (see page 54)	Get the description of current video frame.
	GetVideoStreamInfo (see page 54)	Get the description of video stream which is being decoded.
	IsFormatSupported (see page 55)	
	IsScaleAvailable (see page 55)	



ICC_H264VideoEncoder Interface

	Name	Description
	AddUserData (see page 219)	Adds user data for the subsequent video frame.
	GetH264VideoFrameInfo (see page 220)	


Properties

ICC_H264VideoEncoder Interface

	Name	Description
	ThreadsAffinity (see page 220)	The affinity mask for object's threads. 0 = default (current process) affinity mask

	ThreadsCount (see page 220)	Maximum number of threads which object can use. 0 = automatic.
	ThreadsPriority (see page 220)	The priority for object's threads.

ICC_AVCItraEncoder Interface

	Name	Description
	InitialTimeCode (see page 232)	Specifies the initial timecode for the first frame.

Properties

InitialTimeCode

Specifies the initial timecode for the first frame.

Syntax

```
__property CC_TIMECODE InitialTimeCode;
```

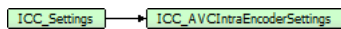
Notes

Should be used before first call to AddFrame ([see page 53](#)).

ICC_AVCItraEncoderSettings Interface

The Settings for AVC-Intra Encoder initialization.



Class Hierarchy





Syntax

```
[object, uuid(54761d8c-4180-46c7-8364-144a64ed1e8e), pointer_default(unique), local]
interface ICC_AVCItraEncoderSettings : ICC_Settings;
```

Properties

	Name	Description
	Mode (see page 233)	The AVC Intra encoder mode.
	NumSingleEncoders (see page 233)	Number of single encoders to be used.

	SequenceHeaderPeriod ( see page 233)	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

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 236)	
	CC_AAC_PROFILE (see page 236)	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 beginning 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 238)	The information about a particular AAC audio frame.
	ICC_AAC_AudioStreamInfo (see page 239)	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

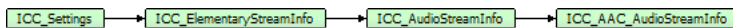
	Name	Description
 R	DTS (see page 28)	The Decoding Data Stamp (DTS) of the elementary data. Usually equals to PTS (see page 28), except the coded video with B-frames.
 R	Duration (see page 28)	Duration of the elementary data. The duration of multimedia samples, encoded into elementary data, measured in CC_TIME units.
 R	NumSamples (see page 28)	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 28)	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 28)	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 28)	The frame's first sample order number.
 R	SequenceEntryFlag (see page 29)	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;
```



Properties

	Name	Description
 R	BitRate (see page 30)	The bitrate (max) of the elementary stream.
 R	FrameRate (see page 30)	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 30)	The elementary stream type. See the CC_ELEMENTARY_STREAM_TYPE (see page 112) for details.

ICC_AudioStreamInfo Interface

	Name	Description
 R	BitsPerSample (see page 87)	The bit depth of one audio sample (of one channel).
 R	ChannelMask (see page 88)	The mask of channels. See the CC_AUDIO_CHANNEL_MASK for details
 R	NumChannels (see page 88)	The number of channels in the waveform-audio data.
 R	SampleRate (see page 88)	The audio sampling frequency.

ICC_AAC_AudioStreamInfo Interface

	Name	Description
 R	Format (? see page 240)	AAC format.
 R	Profile (? see page 240)	AAC profile.

Properties

Format

AAC format.

Syntax

```
__property CC_AAC_FORMAT * Format;
```

Profile



AAC profile.

Syntax

```
__property CC_AAC_PROFILE * Profile;
```

AAC Encoder

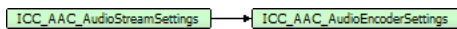
Interfaces

	Name	Description
	ICC_AAC_AudioEncoderSettings (see page 241)	The settings for the CinecoderMpegAudioEncoder initialization.
	ICC_AAC_AudioEncoder (see page 241)	The main interface to access the CC_AAC_AudioEncoder class.

ICC_AAC_AudioEncoderSettings Interface

The settings for the CinecoderMpegAudioEncoder initialization.

Class Hierarchy



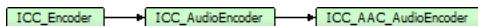
Syntax

```
[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
	GetAudioFrameInfo (see page 84)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 84)	Get the description of audio stream which is being decoded.
	ProcessAudio (see page 85)	Puts another audio uncompressed data to the audio consumer.

AAC Decoder

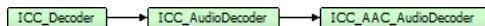
Interfaces

	Name	Description
	ICC_AAC_AudioDecoder (see page 242)	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
	GetAudio (see page 81)	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 82)	Get the description of the current audio frame.
	GetAudioStreamInfo (see page 82)	Get the description of the audio stream.
	GetSampleBytes (see page 83)	
	IsFormatSupported (see page 83)	

Properties

	Name	Description
	NumSamples (see page 83)	The number of ready audio samples at the output.

MP4 Multiplex

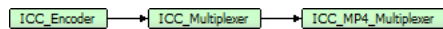
Interfaces

	Name	Description
	ICC_MP4_Multiplexer ( see page 244)	The interface to operate with the MP4 multiplexer.
	ICC_MP4_MultiplexerSettings ( see page 246)	The interface to operate with MP4 multiplexer.
	ICC_MP4_MuxerPinSettings ( see page 248)	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
	CreatePin (see page 98)	Method creates an input pin to add the specified type of elementary data needed to be multiplexed.
	CreatePinByXml (see page 99)	Method creates an input pin to add the specified type of elementary data needed to be multiplexed.
	GetPin (see page 99)	

Properties

	Name	Description
	PinCount (see page 100)	/// Method creates an input pin to add the specified type of elementary data needed to be multiplexed. /// Returns: Returns S_OK if successful or error code otherwise. HRESULT CreatePinByType([in] CC_ELEMENTARY_STREAM_TYPE (see page 112) stream_type, // Pin type [out,retval] ICC_ByteStreamConsumer **pOutput // Address where pointer to the created object will be stored.);

ICC_MP4_Multiplexer Interface

	Name	Description
	Duration (see page 245)	

Properties

Duration

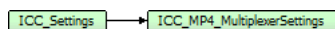
Syntax

```
__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;
```

Properties

	Name	Description
	DataGranularity (see page 246)	Output Block.
	Format (see page 246)	
	Fragmented (see page 247)	
	UpdatePeriod (see page 247)	Index update interval (default is never - only at close).
	WriteSingleFragment (see page 247)	

Properties

DataGranularity

Output Block.

Syntax

```
__property CC_UINT DataGranularity;
```

Format

Syntax

```
__property ICC_Settings * Format;
```

Fragmented

Syntax

```
__property CC_BOOL Fragmented;
```

UpdatePeriod

Index update interval (default is never - only at close).

Syntax

```
_property CC_PERIOD UpdatePeriod;
```

WriteSingleFragment

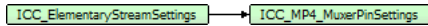
Syntax

```
_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 248)	The time stamp of the first stream sample. Default is 0.
	ReferenceURL (see page 248)	The URL of external (reference) file where the stream will be actually stored (see ICC_ReferenceDataConsumer interface)
	SampleRate (see page 249)	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 249)	Any additional type specification (AVC_Intra type, LPCM format, etc)
	TrackTimeScale (see page 249)	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;
```

Additional codecs

AES-3 (SMPTE-302M) Audio codec

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;

// Ppeneing 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;
}
```