INTERNATIONAL STANDARD

ISO/IEC 23004-2

First edition
2007-10-15

Information technology — Multimedia Middleware —

Part 2: Multimedia application programming interface (API)

Technologies de l'information — Intergiciel multimédia —

Partie 2: Interface de programmation d'application multimédia (API)

Reference number
ISO/IEC 23004-2:2007(E)

© ISO/IEC 2007

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2007

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO/IEC 2007 – All rights reserved

Not for Resale

Contents

Page

Foreword ... iv

Introduction ... v

1 Scope ... 1

2 Organization of this document ... 1

3 Normative references ... 1

4 Terms, definitions and abbreviated terms ... 2

5 Overview of interface suites ... 16
5.1 General interaction ... 16
5.2 Audio ... 17
5.3 Video ... 19
5.4 Security ... 23

6 General interface suites ... 24
6.1 General interaction and navigation ... 24

7 Audio interface suites ... 123
7.1 Broadcast Audio decoding ... 123
7.2 Audio processing and rendering ... 384

8 Video interface suites ... 530
8.1 Broadcast Video decoding ... 530
8.2 Video processing and rendering ... 1041
8.3 Personal video recording ... 1336

9 Security interface suites ... 1395
9.1 IPMP interface suites ... 1395
9.2 Governance interface suites ... 1420

Annex A (normative) **Global types** ... 1437

Annex B (informative) **Use Cases** ... 1449

Annex C (informative) **Application notes** ... 1452

Annex D (informative) **Notational conventions** ... 1453

Annex E (informative) **Generation of channel status information** ... 1455

Annex F (informative) **Application notes** ... 1457

Annex G (informative) **Use of the data Injector in the PVR context** ... 1458

Annex H (informative) **Implementation notes** ... 1459

Bibliography ... 1461

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 23004-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 23004 consists of the following parts, under the general title *Information technology — Multimedia Middleware*:

— *Part 1: Architecture*

— *Part 2: Multimedia application programming interface (API)*

— *Part 3: Component model*

— *Part 4: Resource and quality management*

— *Part 5: Component download*

— *Part 6: Fault management*

— *Part 7: System integrity management*

© ISO/IEC 2007 – All rights reserved

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

MPEG, ISO/IEC JTC 1/SC 29/WG 11, has produced many important standards (MPEG-1, MPEG-2, MPEG-4, MPEG-7, and MPEG-21). MPEG feels that it is important to standardize an application programming interface (API) for Multimedia Middleware (M3W) that complies with the requirements found in the annex to the Multimedia Middleware (M3W) Requirements Document Version 2.0 (ISO/IEC JTC 1/SC 29/WG 11 N 6981).

The objectives of Multimedia Middleware (M3W) are to allow applications to execute multimedia functions with a minimum knowledge of the middleware and to allow applications to trigger updates to the middleware to extend the middleware API. The first goal can be achieved by standardizing the API that the middleware offers. The second goal is much more challenging, as it requires mechanisms to manage the middleware API and to ensure that this functions according to application needs. The second goal can support the first, by reducing the needed standard API to those that provide middleware management. Consequently, applications can use these standard management APIs to generate the multimedia system they require.

ISO/IEC 23004 provides the following:

1. a *vision* for a multimedia middleware API framework to enable the transparent and augmented use of multimedia resources across a wide range of networks and devices;

2. a method to facilitate the integration of APIs to software components and services in order to harmonise *technologies* for the creation, management, manipulation, transport, distribution and consumption of content;

3. a *strategy* for achieving a multimedia API framework by the development of specifications and standards based on well-defined functional requirements through collaboration with other bodies.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Information technology — Multimedia Middleware —

Part 2:
Multimedia application programming interface (API)

1 Scope

This part of ISO/IEC 23004 defines the Multimedia application programming interface (API) of MPEG Multimedia Middleware. The context of this Multimedia API is described in ISO/IEC 23004-1.

2 Organization of this document

This part of ISO/IEC 23004 has the following high level structure:

— Clause 1 defines the scope of this part of ISO/IEC 23004.

— Clause 3 gives an overview of documents that are indispensable for the application of this part of ISO/IEC 23004.

— Clause 4 gives the terms and definitions used in this part of ISO/IEC 23004.

— Clause 5 gives an overview of the interface suites that are part of the Multimedia API of Multimedia Middleware (M3W).

— Clause 6 contains the specification of the general interface suites used for navigation between interfaces, configuration of IO and verification that objects are still alive.

— Clause 7 contains the specification of the interface suites that are part of the Multimedia API and which deal with audio.

— Clause 8 contains the specification of the interface suites that are part of the Multimedia API and which deal with video.

— Clause 9 contains the specification of the interface suites that are part of the Multimedia API and which deal with security and governance.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23004-1, *Information technology — Multimedia Middleware — Part 1: Architecture*

ISO/IEC 23004-3, *Information technology — Multimedia Middleware — Part 3: Component model*

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4 Terms, definitions and abbreviated terms

For the purposes of this document, the following terms, definitions and abbreviated terms apply.

4.1
2CS
Dual or Two Carrier System
German stereo
FM modulated signal which has two carriers, the second of which contains either stereo information or a second (dual) sound channel

4.2
AA
Anti-Aging logical component

4.3
AAC
Advanced Audio Coding (AAC is standardized as part of the MPEG-2 and MPEG-4 specifications.)

4.4
AAVL
Audio Automatic Volume Leveler

4.5
AC-3
Dolby's third generation Audio Coding algorithm a.k.a. Dolby Digital

4.6
access unit
coded representation of a presentation unit

4.6.1
access unit
⟨audio⟩ coded representation of an audio frame

4.6.2
access unit
⟨video⟩ all the coded data for a picture, and any stuffing that follows it, up to but not including the start of the next access unit

4.7
ADRC
Audio Dynamic Range Control logical component

4.8
AFC
Automatic Frequency Control

4.9
AGC
Automatic Gain Control

4.10
AL
Ambient Light

4.11
AM
Amplitude Modulation

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.12
ANG
Audio Noise Generator logical component

4.13
API
Application Programming Interface

4.14
application
piece of software that makes use of the API

4.15
ASK
Amplitude Shift Keying

4.16
ATRAC
Adaptive TRansform Acoustic Coding

4.17
ATSC
Advanced Television Systems Committee

4.18
AtscDec
ATSC Decoder logical component

4.19
AVC
Audio Volume Control logical component

4.20
AVD
Analog Video Decoder

4.21
AVDC
Analog Video Decoder Client

4.22
AVI
Auxiliary Video Information

4.23
AVL
Automatic Volume Leveling

4.24
BBD
Black Bar Detection logical component

4.25
BC
Bass Compensation type of bass enhancement, described in 7.2.2

4.26
BCS
Brightness, Contrast and Saturation

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.27
BE
Bass Enhancement

4.28
BER
Bit Error Rate
ratio of the number of erroneous bits to the total number of bits transmitted

4.29
BTSC
Broadcast Television Systems Committee
US standard for sound, which can be mono or stereo, with an optional SAP

4.30
BVF
Basic Video Featuring

4.31
CA
Conditional Access

4.32
CBC
Cipher Block Chain

4.33
CC
Closed Caption

4.34
CDS
Color Dependent Sharpness

4.35
CHD
Channel Decoding logical component

4.36
CINCH
RCA
connector used for composite video or audio connections

NOTE For stereo audio two connections are needed.

4.37
CLUT
Color LookUp Table

4.38
COM
Microsoft's Component Object Model

4.39
control interface
interface provided by the API that allows platform functionality to be controlled by a client

NOTE In this part of ISO/IEC 23004, the term control interface is in most cases used to mean the interface that the Connection Manager provides to select use cases.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.40
CRC
Cyclic Redundancy Check

4.41
CRT
Cathode Ray Tube

4.42
CTI
Color Transient Improvement

4.43 CVBS

4.43.1
CVBS
Composite Video Baseband Signal

4.43.2
CVBS
Composite Video, Blanking, Synchronization
video format where luminance, chrominance, blanking and synchronization are integrated in one signal

4.44
CVI
Component Video Input
video format where color information is transmitted in three different signals

4.45
CW
Control Word

4.46
DARS
Digital Audio Reference Signal

4.47
dB
deciBel
unit used to express relative difference in power or intensity, usually between two acoustic or electric signals, equal to ten times the common logarithm of the ratio of the two levels

4.48
DBE
Dynamic Bass Enhancement type of bass enhancement, described in 7.2.2

4.49
Dei
De-interlacing

4.50
DES
Data Encryption Standard

4.51
DiSEqC
Digital Satellite Equipment Control

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.52
DLNA
Digital Living Network Alliance

4.53
DMA
Direct Memory Access

4.54
DNR
Dynamic Noise Reduction

4.55
Dolby
Dolby Laboratories Inc., supplier of audio algorithms including the AC-3 audio coding standard

4.56
DRC
Dynamic Range Control
functionality associated with the ADRC

4.57
DSS
Digital Satellite System

4.58
DTS
Digital Theater System

4.59
DTV
Digital TeleVision, including HDTV (high-definition digital television) and SDTV (standard digital television)

4.60
DTVCC
Digital TeleVision Closed Captioning

4.61
DV (SD-2ch/4ch)
Digital Video (IEC 61834) interface standard

4.62
DVB
Digital Video Broadcasting

4.63
DVD
Digital Versatile Disc

4.64
DVI
Digital Visual Interface

4.65
ECB
Electronic CodeBook

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.66
ECM
Entitlement Control Message

4.67
EIAJ
Electronic Industries Association of Japan
Japanese TV stereo sound standard

4.68
EIT
Event Information Table

4.69
EMM
Entitlement Management Message

4.70
error
condition which, if not handled, can cause a system to crash or malfunction

4.71
error handler
piece of software meant to deal with the consequences of an error

4.72
ES
Elementary Stream

4.73
EXIF
EXchangeable Image File format

4.74
FEC
Forward Error Correction

4.75 Field/frame rate

4.75.1
field rate
⟨interlaced video⟩ number of fields per second

4.75.2
frame rate
⟨progressive video⟩ number of frames per second

4.76
filter depth
size of the data on which filter criteria can be applied

4.77
FM
Frequency Modulation

4.78
FM Korea
form of 2CS transmission used in South Korea

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.79
GIF
Graphics Interchange Format from CompuServe

4.80
GOP
Group Of Pictures

4.81
GUID
globally unique identifier
value that uniquely identifies some entity in the universe

4.82
HDMI
High-Definition Multimedia Interface
uncompressed digital audio/video interface

4.83
HM
Histogram Modification

4.84
IAA
I Am Alive logical component

4.85
ICAM
Integrated Conditional Access Module

4.86
ID
Identifier

4.87
Id
Identification

4.88
IF
Intermediate Frequency

4.89
IID
Interface Identifier

4.90
JFIF
JPEG File Interchange Format

4.91
JPEG
Joint Photographic Experts Group

4.92
kbps
kilobits per second

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.93
LC
Logical Component
multimedia component as it appears to the user

4.94
LCD
Liquid Crystal Display

4.95
LFE
Low Frequency Effects

4.96
LNB
Low Noise Block

4.97
loudness
perceived strength of sound

NOTE Loudness generally increases with the volume level, but not uniformly across all frequency ranges for the human ear.

4.98
Loudness-Processing
advanced audio functionality that ensures that the relative perceptual loudness levels for different frequencies are maintained across volume settings for the same musical piece

4.99
Lt/Rt audio stream
two-channel (Dolby, DTS, etc.) surround encoded audio stream

4.100
LTI
Luminance Transient Improvement

4.101
matrix display
flat display type, like plasma or LCD

4.102
McTc
Motion compensated Temporal conversion

4.103
MPEG
Moving Picture Experts Group

4.104
MPEG section
segment of tables containing meta-data

NOTE See [2] for more information.

4.105
MS
MPEG Source

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.106
MSB
Most Significant Bit

4.107
MTS
Multichannel Television Sound standard
standard similar to FM radio stereo modulation techniques in conjunction with an additional SAP carrier

4.108
multi-channel signal
audio signal with at least one rear channel (a.k.a. surround channel) and/or at least three front channels

4.109
multiplexing
⟨TS Mux⟩ inserting one or more section data packets into a stream of packets(TS/TSts) at appropriate points in the stream

4.110
NG
Noise Generation
functionality associated with the ANG

4.111
NICAM
Near Instantaneously Compacted Audio Multiplex
digital sound standard which can contain mono sound, stereo sound, dual sound or data, which is either related or unrelated to the image

NOTE NICAM is accompanied by an AM or FM mono signal.

4.112
NIT
Network Information Table

4.113
NTSC
National Television Standards Committee

4.114
Null use case
special use case that consists of zero streaming graphs and hence has no components

NOTE This use case is always the initial use case after a Connection Manager has been instantiated.

4.115
OFB
Output FeedBack

4.116
OFDM
Orthogonal Frequency Division Multiplex

4.117
OOBTD
Out Of Band Tuning and Demodulation logical component

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.118
PAL
Phase Alternating Line

4.119
PAT
Program Association Table

4.120
PB
Phantom Bass type of bass enhancement, described in 7.2.2

4.121
PCM
Pulse Code Modulation
digital mono or stereo audio stream

4.122
PCR
Program Clock Reference

4.123
PES
Packetised Elementary Stream

4.124
PID
Packet IDentifier

4.125
platform
collection of components of which multiple platform instances can be instantiated

4.126
platform instance
instantiation of a platform, which determines which components are used from the platform (and how many of each), how they are configured and which use cases and possible transitions between them are possible

4.127
PMT
Program Mapping Table

4.128
PNG
Portable Network Graphics

4.129
program
digital video broadcasting service

4.130
PS
Pseudo-Sync

4.131
PSI
Program Specific Information

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.132
PSK
Phase Shift Keying

4.133
PTS
Presentation Time Stamp

4.134
PVR
Personal Video Recorder

4.135
QAM
Quadrature Amplitude Modulation
modulation of two orthogonal derivates of a carrier by two signals

4.136
QPSK
Quadrature Phase Shift Keying

4.137
RF
Radio Frequency

4.138
RFA
RF Amplification logical component

4.139
RGB
component video format where color information is transmitted in a red, green and blue signal

4.140
SAP
Second Audio Program, used with the BTSC sound standard and always mono

4.141
Scan Rate
combination of frequency and scan type

4.142
SCART
Peritel
Euroconnector
connector used for combined audio and video connections

NOTE 1 A formal description is given in CENELEC EN 50 049-1:1989 and in IEC 933-1.

NOTE 2 SCART is the acronym for *Syndicat des Constructeurs d'Appareils Radiorécepteurs et Téléviseurs*.

4.143
SDC
System time clock Decoder Client

4.144
SE
Sharpness Enhancement, based on LTI and peaking

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.145
SECAM
Séquentiel Couleur Avec Mémoire or *Système Electronique Couleur Avec Mémoire*, translated as SEquential Color And Memory

4.146
SI
Service Information

4.147
SPD
Source Product Description

4.148
SPDIF
Sony/Philips Digital InterFace
standard audio transfer file format over a standard CINCH or optical connector

4.149
SPIFF
Still Picture Interchange File Format

4.150
SPKR
speaker-set headphones logical component

4.151
SPTS
Single Program Transport Stream

4.152
SRS
Sound Retrieval System

4.153
STC
System Time Clock

4.154
STCD
STC Decoder

4.155
streaming graph
collection of one or more components that are connected via their inputs and outputs and that stream data in between them

4.156
TDES
Triple DES

4.157
thread
single thread of execution

4.158
thread-local storage
mechanism by which variables can be allocated such that there is one instance of the variable per extant thread

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE Most operating systems provide special functions to allocate and access such thread-specific variables.

4.159
thumbnail
small image representation of a larger image, usually intended to make it easier and faster to look at or manage a group of larger images

4.160
TIFF
Tag Image File Format

4.161
TP
Transport Packet

4.162
TPH
Transport Packet Header

4.163
TS
Transport Stream

4.164
TS Demux
Transport Stream Demultiplexer

4.165
TS Mux
Transport Stream Multiplexer

4.166
TSts
Transport Stream with time-stamps

EXAMPLE a MPEG-2 transport stream having a 32-bit time-stamp appended to every transport stream packet

4.167
M3W client
object that uses a M3W interface

NOTE Such an object does not necessarily provide the uhIUnkown interface, unless it provides a M3W interface itself, such as a notification interface.

4.168
M3W object
object that provides a M3W interface

NOTE Such an object has to provide the uhIUnknown interface as well.

4.169
URL
Uniform Resource Locator

NOTE Previously known as Universal Resource Locator.

4.170
use case
collection of zero or more streaming graphs

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE By selecting a use case, the client determines which components are active and how the connections between those components are laid.

4.171
UV
color component of a YUV signal

4.172
VBI
Vertical Blanking Interval

4.173
VCE
Video Color Enhancement logical component

4.174
VCR
Video Cassette Recorder

4.175
VPS
Video Programming System

4.176
VS
Vendor Specific

4.177
VSB
Vestigial SideBand

4.178
WSS
Wide Screen Signalling

4.179
WST
World Standard Teletext

4.180
Y/C
video format where color information is transmitted in a luminance (Y) and one chrominance (C) signal

4.181
Y/CVBS
luminance/Composite Video Blanking Signal

4.182
Y'CbCr
color format using a luma (gamma-corrected luminance) and two chroma (gamma-corrected chrominance) components

4.183
YC
video format where separate luma (Y) and chroma (C) video signals are used, rather than a single composite video signal

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.184
YCbCr
digital component video format where color information is transmitted in a luminance (Y) and two color (Cb and Cr) signals

NOTE YCbCr is a digital version of YPbPr.

4.185
YPbPr
component video format where color information is transmitted in a luminance (Y) and two color (Pb and Pr) signals

NOTE YPbPr is a scaled version of YUV.

4.186
YUV
component video format where color information is transmitted in a luminance (Y) and two color (U and V) signals

4.187
YUV signal
video signal composed of a luminance component (Y) and a chrominance component (UV)

5 Overview of interface suites

5.1 General interaction

This clause is informative.

In this subclause, the individual specifications with their name and a brief description of the addressed functionality in the specification are listed.

Table 1 — Description of the general interface specifications

Specification	Name	Description
uhIConnMgr	Connection Manager	This specification is special in the sense that the interfaces of the Connection Management interface suite are templates, which are instantiated for each platform instance. This is because the interfaces are used to select the current use case and reporting completion of the selection, and the set of use cases is specific to each platform instance.
uhIFatErr	Fatal Error	This API specification defines the interfaces associated with the fatal and non-fatal error handling.
uhIIamAlive	I am alive	This API specification defines the I am Alive interface suite which consists of the interfaces uhIIamAlive and uhIIamAliveNtf, that support “watchdog” functionality.
uhIOutConf	Output Configuration	Component represents functionality to select an output configuration for the audio input stream, i.e. select how and with how many channels the audio input stream should be rendered.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

uhIPinObjects	Pin Objects	This suite contains a Pin Objects Provider that has functionality to give access to pin-related objects. Pins refer to the inputs and outputs of a logical component. The Pin Object Provider is meant to be a template only and has to be specialized by logical components that have pin-related objects.
rcIUnknown / uhIUnknown	IUnknown	This API specification defines the rcIUnknown interface which is the M3W equivalent of the IUnknown interface of Microsoft COM. The rcIUnknown interface is a generic interface that is provided by all objects and inherited by all other interfaces. It has three main functions: — providing interface navigation functionality; — providing reference counting functionality; — defining the identity of an object.

5.2 Audio

In this subclause, the individual specifications with their name and a brief description of the addressed functionality in the specification are listed.

Table 2 — Description of the broadcast audio decoding specifications

Specification	Name	Description
uhIAnaAdec	Analog Audio Decoding	Decodes an encoded analog audio input stream, and outputs the decoded programs. Determines the sound standard of the incoming stream from a set of allowed sound standards and provides notifications of various changes that occur in the input stream.
uhIAtscDec	ATSC Decoder	This component is responsible for demultiplexing and decoding an ATSC compliant Transport Stream (TS) into its constituent components.
uhISpdifDec	SPDIF Decoding	Decodes an originally SPDIF-formatted encoded audio input stream, and outputs a decoded audio stream. Determines the compression standard of the encoded audio incoming stream and provides notifications of various changes that occur in the input stream.
uhISpdifIn	SPDIF-in	Parses the data from a digital input signal that complies to the SPDIF format. The parsed meta data is available to the client. Both meta data and the audio data (linear PCM or encoded audio) are available at the output.
uhISpdifOut	SPDIF-out	Transforms various formats of a digital audio stream to a SPDIF IEC compliant stream.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 3 — Description of the audio Processing & Rendering

Specification	Name	Description
uhlAavl	Audio Automatic Volume Levelling	Compensates for the variation in the volume level, that are e.g. caused by a change in the audio content (advertisements) or a change in the source of the incoming audio signal, resulting in less annoyance for the end-user.
uhlAbassEnh	Audio Bass Enhancement	Provides control on a set of signal processing algorithms that improve listening experience for low-frequency (bass) sounds. An appropriate choice of the algorithm depends on the kind of bass rendering capabilities the system speakers have.
uhlAdrc	Audio Dynamic Range Control	Reduces the dynamic range of the audio incoming signal to get low-level and high-level signals closer together in level, the incoming signal can thus be adapted to the reproduction apparatus and/or the listening conditions.
uhlAmix	Audio Mixing	Mixes a number of incoming (uncompressed, baseband) audio streams into a single ouput audio stream; individual control is provided to mute, unmute, and control the gain of each of the incoming streams.
uhlAnoiseGen	Audio Noise Generation	Provides control on the injection of noise into audio channels to be fed to a set of speakers (noise injected a channel at a time), enabling the calibration of the inter-channel relative reproduction levels at the listening position.
uhlAprogSel	Audio Program Selection	Selects one (or more) audio program (e.g. a language) from an audio incoming stream with multiple programs.
uhlAvolCtrl	Audio Volume Control	Provides control on the sound level of the audio signal outputted by the audio playback system (master volume); also provides control on muting of the audio signal, and on Loudness processing.
uhlEqualizer	Equalizing	Provides control to adjust the signal level of the audio incoming signal in various frequency bands.
uhlOutConf	Output Configuration	Provides control to a set of algorithms to get an audio stream with a specific output configuration from an audio incoming stream: selection among a set of possible algorithms covering surround processing, virtualizing, downmixing and upmixing processes and selection of the number of channels in the output stream.
UhlSpkr	Speaker Set/Headphones	Activates the speakers to be used in the system and provides control to bass-redirection, trim control and delay functionalities.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.3 Video

In this section the individual specifications with their name and a brief description of the addressed functionality in the specification are listed.

Table 4 — Description of the broadcast decoding specifications

Specification	Name	Description
UhlAnaAvDem	Analog Audio & Video Demodulation	Demodulate the signal, and output the resulting Y/CVBS video and encoded audio signal. Determines the TV system of the incoming signal so the sound carrier can be located.
UhlAnaVdec	Analog Video Decoding	Decode a composite video stream (e.g. CVBS, Y/C) into a component decoded signal (e.g. YUV) or to decode a component video stream into another type of component video stream. It also notifies properties of the decoded video signal.
UhlAtscDec	ATSC Decoder	This component is responsible for demultiplexing and decoding an ATSC compliant Transport Stream (TS) into its constituent components.
UhlAvIn	Analog AV Input	Detect base band signal properties associated with an AV input cluster (e.g. a SCART) and notifies changes of these properties.
UhlAvOut	Analog AV Output	Mute audio and video outputs and signal base band signal properties associated with an AV input cluster (e.g. a SCART).
UhlChanDec	Channel Decoding	Control digital demodulation for DVB cable (DvbC), terrestrial (DvbT), and satellite (DvbS) reception.
UhlHdmiIn	HdmiIn	Extracts info frames from a HDMI stream and passes them on to the client.
uhlImageDec2	Image Decoding 2	Decode full images to a requested format, get thumbnails in a requested format, and extract raw meta-data.
UhlOob	Out Of Band Tuning & Demodulation	Receive an external out of band signal, tune to the correct frequency and demodulate that signal. This is typically used by clients that adhere to the OpenCable POD host control interface standard.
UhlRfAmp	RF Amplification	Control the amplification of an incoming RF signal in order to optimize the signal processing by a tuner. Typically needed in regions where transmissions are weak, over modulated, or heavily distorted.
UhlSigStrength	Signal Strength	represents functionality to measure the strength of an incoming IF signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

UhlStcDec	STC Decoder	Provide a constant time base for synchronized presentation of components (audio, video, subtitles, etc.) of a digital MPEG stream. The STCD synthesizes the STC from time stamps that have been extracted from an MPEG stream (in push mode), or it bases the STC on an autonomously running clock that is linked to, but not driven by, the PTS values in the decoded video or audio elementary stream (in pull mode).
UhlTsDmx	Transport Stream Demultiplexing	TS Demux represents functionality to filter specific content (e.g. PES packets, sections) from a transport stream (TS).
UhlTsMux	Transport Stream Multiplexing	Control the rate and number of insertions per section into this transport stream and output this multiplexed transport stream.
UhlTuning	Tuning	Tune to a specific frequency (for e.g.. an antenna, cable or satellite dish). It can perform many operations like tuning, fine-tuning and optionally Automatic Frequency Control (AFC) and searching on the input signal.
UhlUrlSrc	URL Source	A URL Source is a streaming source that streams data identified by a URL into the platform (based on protocols like e.g. http, ftp, rtsp, file etc.).
uhlVbiSlice	VBI Slicing	Extract Vertical Blanking Interval data from an analog video stream.

In Figure 1 an example graph is shown based on the interface specification listed for decoding a digital and an analog input signal.

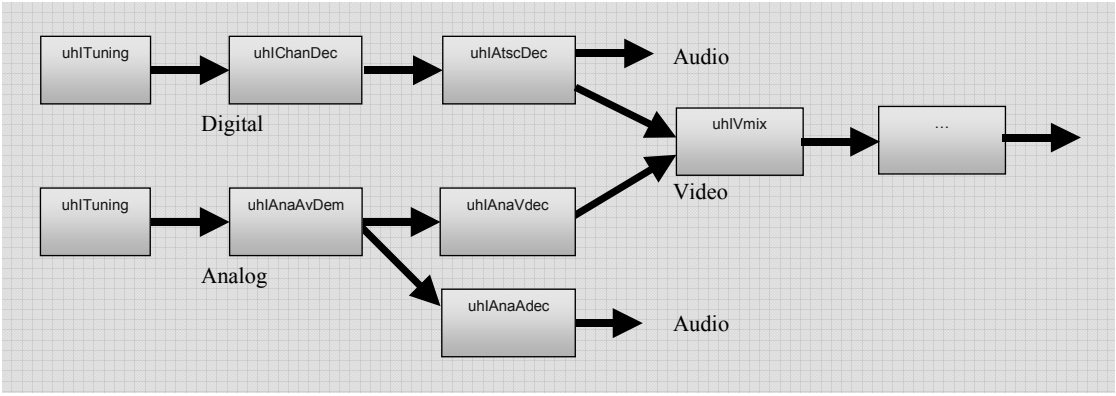


Figure 1 — Simplified example graph.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 5 — Description of the video Processing & Rendering.

Specification	Name	Description
uhIAmbientLevel	Ambient Level	Set the measured ambient light. This can then be used by the platform to optimise the video processing algorithms based on this.
UhIAnaVenc	Analog Video Encoder	The Analog Video Encoder gives control over the encoding of a digital video stream into an analog video signal.
uhIAnaVencrypt	Analog Video Encryption	The Analog Video Encryptor gives control over an anti-taping feature.
uhIAntiAging	Anti Aging	Control the algorithm used to prevent aging in case of static images being displayed (e.g. logo's).
UhIBbarDet	Black Bar Detection	Detection of the position of the black bars in the video signal. These are often present if the transmission uses a different aspect ratio then the original recording.
UhIColEnh	Video Color Enhancement	Enhance the colors of the incoming video signal. Color enhancement deals with the color features Skin Tone Correction, Blue Stretch, and Green Enhancement.
UhICti	Color Transient Improvement	Allows control over the color transient improvement functionality (to increase the perceived color sharpness).
UhIDnr	Dynamic Noise Reduction	To control the reduction of the noise in a video stream.
uhIHistoMod	Histogram Modification	Histogram Modification can be used to improve the local contrast in a picture of which the distribution of grey levels is sub-optimal.
uhINoiseMeas	Noise Measurement	The Noise Measurement offers an abstract view on the amount of noise in a video signal.
uhIScanRateConv2	Scan Rate Conversion 2	Scan Rate Conversion provides the control on different algorithms to change the scan rate and or scan type of a signal (progressive, interlaced).
uhISharpEnh	Sharpness Enhancement	Sharpness Enhancement deals with enhancing the sharpness of the luminance (Y) component of a YUV signal.
uhISharpMeas	Sharpness Measurement	The Sharpness Measurement logical component has a YUV signal as input. It measures the sharpness of the Y signal (luma sharpness) and the sharpness of the UV signal (color sharpness).
UhIVfeat	Basic Video Featuring	Basic Video Featuring represents functionality to control the brightness, contrast, saturation, hue, and white point of a video signal.
UhIVmix	Video Mixing	Video Mixer represents functionality to compose a single video stream from multiple inputs, each of which can either be video or graphics.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In Figure 2 an example graph is shown based on the interface specifications listed in Table 5.

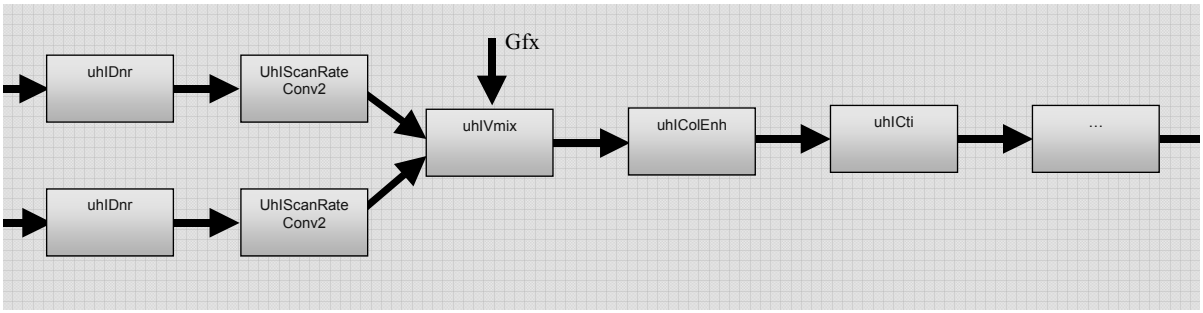


Figure 2 — Possible Video processing & Rendering graph.

Table 6 — Description of the personal Video Recording related.

Specification	Name	Description
uhIDataInject	Data Injecting	The Data Injecting component allows the injection of streaming data into the platform. This can be used for a PVR application.
uhIDataExtract	Data Extracting	The Data Extracting component allows the extraction of streaming data from the platform. This can be used for a PVR application.
uhISptsTrMux	SPTS Transmuxing	Provides functionality to transmux an incoming Transport Stream to a SPTS, to insert sections in this SPTS and to notify clients of the locations of video frames in the SPTS output.

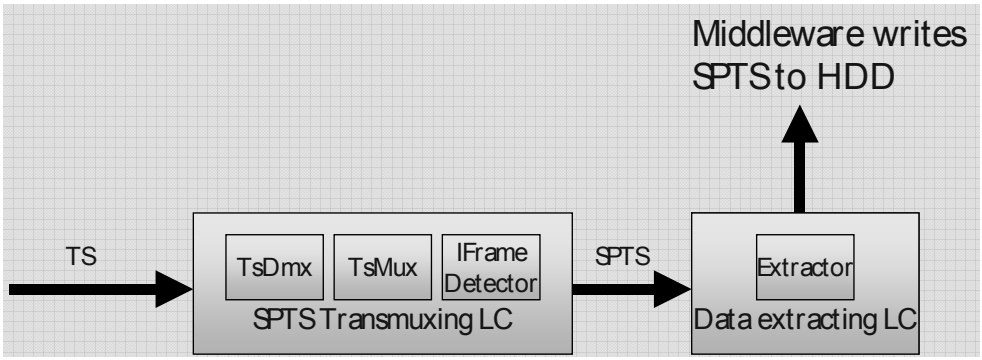


Figure 3 — Extracting a Single Program Transport Stream for recording.

In Figure 3 a simplified context is given. Here the SPTS Transmux is used to reduce the bandwidth, by selecting the specific stream to be recorded, of the incoming stream. After this bandwidth reduction, the stream can be recorded to any device that is typically supported by the operating system (e.g. a harddisk). Obviously in a real product there will be a tuner and channel decoder etc. in front of the SPTS Transmux from Table 6.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

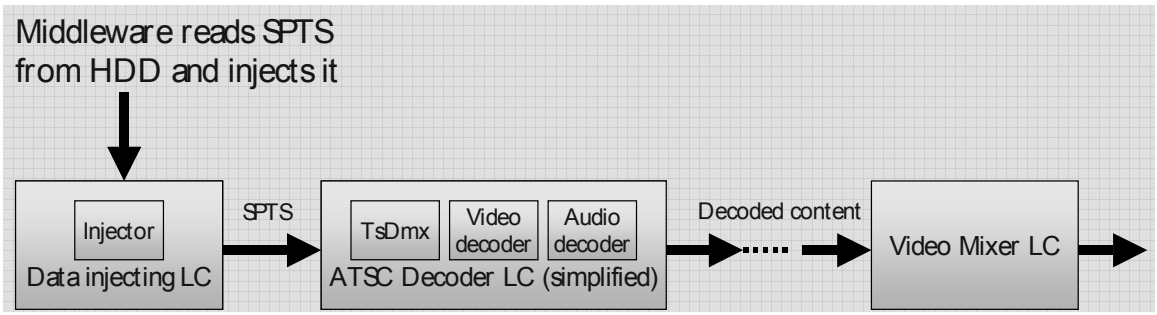


Figure 4 — Injecting a Single Program Transport Steam for playback.

In Figure 4 a simplified context is shown for the Data Injector to show how to play back a recorded stream. The normal Atsc Decoder can be used to decode the recorded stream. The AtscDec provides the controls for the trickplay functionality (e.g. control playback speed). Obviously in a real product more components will be there like e.g. Video Processing and Rendering components (see Table 5).

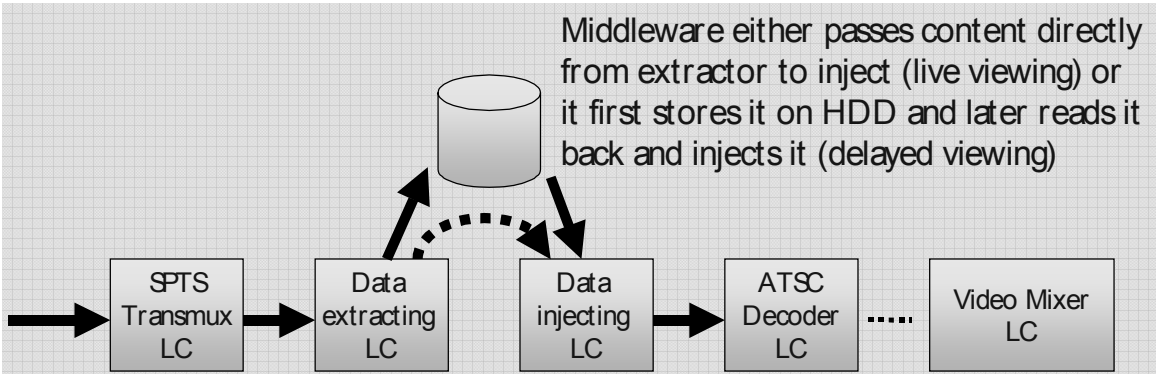


Figure 5 — Combining Data Extraction and Injection for delayed viewing.

In Figure 5 a simplified combined usage is shown from the SPTS Transmux, the Data Extractor and the Data Injector interface specifications. In this case a watch delayed video use case is given. A real product obviously contains more interface specification like e.g. a Tuner, Channel Decoder and Video Processing and Rendering components (see Table 5).

5.4 Security

In this subclause, the individual specifications with their name and a brief description of the addressed functionality in the specification are listed.

Table 7 — Description IPMP related.

Specification	Name	Description
Trust Management	Trust Management interfaces	In trust management interfaces, it needs to consider how to access secured or governed information in processing multimedia data or performing IPMP tools.
IPMP Tool	IPMP Tool	During the processing of MPEG-2/4, whenever and wherever IPMP processing is required, IPMP tools perform their own IPMP functions.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 8 — Description Governance related.

Specification	Name	Description
IGovernance	Governance	This interface suite deals with restricting access to realization elements.

6 General interface suites

6.1 General interaction and navigation

6.1.1 uhIConnMgr

6.1.1.1 Concepts

6.1.1.1.1 Use Cases

The platform consists of streaming components (like digitizers, filters and renderers) that can be connected to each other to form all kinds of configurations, which are used to build multi-media applications. Dynamic switching between configurations is required because:

— the memory footprint can be reduced when at each moment only those components are instantiated that are actually used.

— the connections between components do not have to be fixed over time. Example: when viewing a digital video stream the demultiplexer is connected to a video decoder component, when recording the stream the demultiplexer has to be also connected to a hard disk writer component.

Typically, the configuration that is needed at a certain moment depends on what usage the application requires at that moment. We will use the term use case to indicate such a particular configuration.

Each platform instance will provide a set of use cases that are available to applications that are built on top of that platform. The application controls which use case is selected at any moment in time. The provided use cases should be such that they are useful to the applications and therefore, have to be developed closely coupled to an application domain.

6.1.1.1.2 Connection Manager

The Connection Manager is the entity that is responsible for performing the actual switching between use cases. It provides a control interface that applications can use to select the appropriate use case. Since each platform instance provides its own set of use cases, this control interface will be specific to each platform instance. The control interface specified in this document should therefore be seen as a “template” which has to be used when specifying the control interface for a platform instance. The following “parameters” of the control interface template have to be specified for each platform instance:

— the set of component instance ids that are part of the platform instance (see subclause 6.1.1.2.1.2).

— the set of use cases provided by the platform instance (see subclause 6.1.1.2.1.3).

— the set of functions for selection of a use case (see subclause 6.1.1.5.1.3). For each use case (or group of related use cases) a separate function has to be specified (see subclause 6.1.1.1.4).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.1.3 Switching between Use Cases

The basic task of the Connection Manager is to perform the switch from the current to a requested use case. The Connection Manager has to make the switch as smooth/seamless as possible. To be more precise, it will

— not destroy and re-create a component instance that is part of both the current and the requested use case (i.e. the component instance keeps its state),

— mute the output signal(s) to prevent visual and audible artifacts if needed.

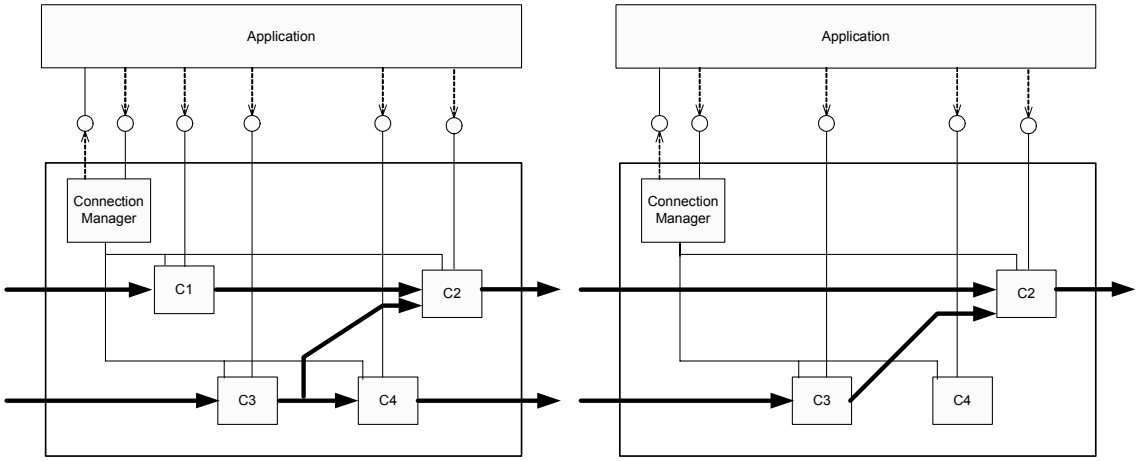


Figure 6 — Example: Switch between two use cases

Figure 6 shows an example of two use cases. *C1* to *C4* are component instances of the platform instance. Streamed data is indicated by the thick lines (arrows indicate the direction.) Component instance *C1* is only part of the left use case and will be stopped and destroyed when switching to the right use case. Component instance *C4* is part of both use cases, but in the second use case it is not connected to any other component. This is allowed to ensure that component instance *C4* keeps its state when switching from the left to the right use case and back, provided that component *C4* allows such behavior.

In principle, a platform instance supports all possible transitions between the use cases. However, it is allowed for a platform instance to exclude certain transitions.

6.1.1.1.4 Functions versus parameters

In principle, the `uhIConnMgrPlfSelUseCase` interface contains for each use case a separate function. This may possibly lead to a huge number of functions. The number of functions can be decreased by grouping similar use cases into one use case by using parameters. For example in a specific platform instance there may be three SCART outputs. By using a parameter in each use case that indicates which SCART output has to be used, the number of functions is effectively divided by three.

Use cases can be grouped into one use case with parameters when the sets of component instances provided by those use cases are the same.

6.1.1.1.5 Retrieving Interfaces

Apart from switching between use cases, the Connection Manager is also the entry point for the application to retrieve interfaces to the component instances that are part of the current use case. Note that only interfaces can be retrieved of component instances that are part of the currently selected use case. (In the right use case of Figure 6 it is not possible to get the an interface of component *C1*). Once an application has retrieved an interface to a component, it can retrieve other interfaces to that component using `QueryInterface` [Annex B of ISO/IEC 23004-1]. (Note that instead of using `QueryInterface` it is also possible to ask the Connection Manager for this interface.)

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When a component creates sub components itself (e.g. a Mixer creates layer components) interfaces to these sub components can only be retrieved via the interface of the main component, not via the Connection Manager.

From the above it becomes clear that an application can only use interfaces of component instances that are part of the current use case. This puts an important constraint on the application: **before selecting a new use case, the application has to release all interfaces that are provided by components (and their sub components) that will not be part of the new use case**. There is no need to release an interface when the component that provides it, is also part of the new use case. (In the example of Figure 6 when switching from the left to the right use case, only the interfaces of component *C1* have to be released.)

The `GetContainedComponents` function can be used to find out which component instances are part of a specific use case.

6.1.1.1.6 Multiple component instances of the same type

A platform instance can contain multiple component instances of the same component type, for example multiple demultiplexers. An application may want to specify from which component instance it requires an interface, especially when the capabilities of the component instances differ (e.g. the number of section filters a demultiplexer supports) or when their connections differ (e.g. to video decoder or hard disk writer). Each component instance therefore gets a unique *component id* which has to be used when requesting interfaces from the Connection Manager. The platform instance documentation will describe which component instances are part of the platform instance and specify their unique component id.

6.1.1.2 Types & Constants

6.1.1.2.1 Public Types & Constants

6.1.1.2.1.1 Error Codes

Signature

```
const uhErrorCode_t uh_ERR_PLF_NO_INTERFACE = CID_PLF | 0x801;
```

Qualifiers

— error-codes

Description

Defines the error codes that can be returned by the functions in the interface(s) of a Connection Manager. Note that each platform specific instantiation of this interface will have to define this error code based on its component id (*CID_PLF*).

Constants

Table 9

Name	Description
`uh_ERR_PLF_NO_INTERFACE`	Requested interface is not supported by the specified component. This can be returned by the `GetInstance` function.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.2.1.2 uhConnMgrPlf_CompId_t

Signature

```
typedef enum _uhConnMgrPlf_CompId_t {
    uhConnMgrPlf_CComp0 = 0,
    uhConnMgrPlf_CComp1 = 1,
...
} uhConnMgrPlf_CompId_t, *puhConnMgrPlf_CompId_t;
```

Qualifiers
None.

Description
This enumeration serves as a generic example. Each platform instance will define its own enumeration.

This enumeration lists the valid component instances IDs. Values of this type can be passed as parameter to the `GetInstance` function.

Values

Table 10

Name	Description
`uhConnMgrPlf_CComp0`	ID of component 0
`uhConnMgrPlf_CComp1`	ID of component 1
`...`	etc.

Remarks
— 'Plf' in the type name, is the platform name and should be filled in by the platform instance designer.

— Each platform instance will at least contain one component (instance).

6.1.1.2.1.3 uhConnMgrPlf_UseCaseId_t

Signature

```
typedef enum _uhConnMgrPlf_UseCaseId_t {
    uhConnMgrPlf_UcNone        = 0,
    uhConnMgrPlf_UcUseCase1    = 1,
...
} uhConnMgrPlf_UseCaseId_t, *puhConnMgrPlf_UseCaseId_t;
```

Qualifiers
None

Description
This enumeration serves as a generic example. Each platform instance will define its own enumeration. However, the first value is always uh`ConnMgrPlf_UcNone`.

This enumeration lists the valid use case IDs. Values of this type have to be passed as parameter to the `GetContainedComponents` function. Furthermore, this type is used by the `OnUseCaseSelected` function to indicate which use case has been selected.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 11

Name	Description
uhConnMgrPlf_UcNone	ID of the Null use case
uhConnMgrPlf_UcUseCase1	ID of use case 1
...	etc.

Remarks

— 'Plf' in the type name, is the platform name and should be filled in by the platform instance designer.

— Each platform instance will at least contain two use cases: the Null use case and a real one.

— There should be a uhConnMgrPlf_UseCaseId_t value for each use case provided by the platform instance.

6.1.1.2.1.4 uhConnMgrPlf_Notification_t

Signature

```
typedef enum _uhConnMgrPlf_Notification_t {
    uhConnMgrPlf_SubscriptionChanged = 0x00000001,
    uhConnMgrPlf_UseCaseSelected     = 0x00000002
} uhConnMgrPlf_Notification_t, *puhConnMgrPlf_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a on-to-one correspondence between these values and the functions in the notification interface uhIConnMgrPlfSelUseCaseNtf.

Values

Table 12

Name	Description
uhConnMgrPlf_SubscriptionChanged	Represents OnSubscriptionChanged.
uhConnMgrPlf_UseCaseSelected	Represents OnUseCaseSelected.

6.1.1.2.1.5 uhConnMgrPlf_Notifications_t

Signature

```
typedef Int32 uhConnMgrPlf_Notifications_t, *puhConnMgrPlf_Notifications_t;
```

Qualifiers

— enum-set

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`ConnMgrPlf_Notification_t`. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

6.1.1.2.1.6 uhConnMgrPlf_AllNotifications_t

Signature
```
const uhConnMgrPlf_Notifications_t uhConnMgrPlf_AllNotifications =
     (uhConnMgrPlf_SubscriptionChanged | uhConnMgrPlf_UseCaseSelected);
```

Qualifiers
None

Description
Defines the set of all values of type uh`ConnMgrPlf_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

6.1.1.2.2 Model Types & Constants

6.1.1.2.2.1 UseCaseSelectionState

Signature
```
enum {
    UseCaseSelectionInProgress = 0,
    UseCaseSelectionCompleted  = 1
} UseCaseSelectionState;
```

Qualifiers
None

Description
This type indicates whether a use case selection is in progress or not. It is used in the pre- and post-conditions of the functions of the `uhIConnMgrPlfSelUseCase` interface.

Values

Table 13

Name	Description
`UseCaseSelectionInProgress`	Selection of a use case is in progress.
`UseCaseSelectionCompleted`	Selection of a use case is completed.

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.2.2.2 Component

Signature

```
entity Component {
    const uhConnMgr_CompId_t cid;
    const setof{uhIid_t}     iidSet;
}
```

Qualifiers

None.

Description

Entities of this type are used to model the component instances that are part of the platform instance. Each component instance has a unique ID and can provide interfaces for each iid that is part of the `iidSet`.

Independent Attributes

Table 14

Name	Description
`cid`	Identifies the component instance uniquely within the platform instance.
`iidSet`	The set of IIDs of the interfaces provided by the component instance, determining for which IIDs the `GetInstance` function will be successful. The value of `iidSet` is constant because it is set at component instantiation time.

Constraints

None.

6.1.1.2.2.3 UseCase

Signature

```
entity UseCase {
    const uhConnMgrPlf_UseCaseId_t id;
    const setof{Component}         compSet;
}
```

Qualifiers

None.

Description

Entities of this type are used to model the use cases that can be selected using the `uhIConnMgrPlfSelUseCase` interface. Each use case has a set of component instances. This set is constant because it is not possible to dynamically add/remove component instances from a use case.

Independent Attributes

Table 15

Name	Description
`id`	Identification of a use case.
`compSet`	Set of component instances that are part of the use case.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Constraints
None.

Remarks
— The connections in between the component instances are not modeled as part of the `UseCase` model type. They are not directly observable by the client.

— Most use cases will also contain some specific data attributes (related to the parameters of the `SelectUseCaseX` function.) This has not been modeled explicitly.

6.1.1.2.2.4 Transition

Signature
```
entity Transition {
    const UseCase from;
    const UseCase to;
}
```

Qualifiers
None.

Description
In some platform instances not all transitions between the use cases may be allowed. Entities of this type are used to model the allowed transitions between use cases.

Independent Attributes

Table 16

Name	Description
`from`	Use case from which the transition should be made (i.e. the use case that is current when the transition is started).
`to`	Use case to which the transition should be made (i.e. the selected use case).

Constraints
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.3 Logical Component

6.1.1.3.1 Interface-Role Model

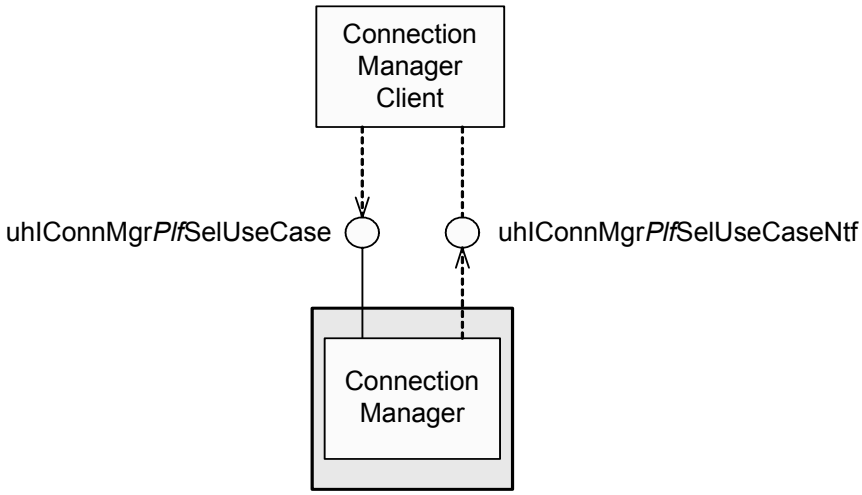


Figure 7 — Interface Role Model

The `ConnectionManager` role provides facilities to select a use case and to retrieve interfaces to the component instances of the selected use case through its `uhIConnMgrPlfSelUseCase` interface. It can send notifications when the use case selection has been completed.

The `ConnectionManagerClient` role can use this interface to subscribe to these notifications by providing a `uhIConnMgrPlfSelUseCaseNtf` interface. In this document we also use the terms *client* and *application* as a shorthand for this role.

6.1.1.3.2 Diversity

6.1.1.3.2.1 Provided Interfaces

Table 17

Role	Interface	Presence
`ConnectionManager`	`uhIConnMgrPlfSelUseCase`	mandatory
`ConnectionManagerClient`	`uhIConnMgrPlfSelUseCaseNtf`	optional[1]

[1] The `uhIConnMgrPlfSelUseCaseNtf` interface is optional, but in practice the client cannot do anything with the `uhIConnMgrPlfSelUseCase` interface if it does not provide this notification interface, unless the function `SelectUseCaseX` is implemented as a synchronous function for the platform instance.

-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.3.2.2 Configurable Items

Table 18

Role	Attribute
`ConnectionManager`	`useCases`
`ConnectionManager`	allowedTransitions

For a description of the attributes see 6.1.1.4 Roles.

6.1.1.3.2.3 Constraints

— For all transition in `allowedTransitions`:

  — transition.to in `useCases`

  — transition.from in `useCases`

6.1.1.3.3 Instantiation

6.1.1.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 19

Type	Object	Multiplicity
`ConnectionManager`	`connmgr`	1

6.1.1.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
connmgr.currentUseCase == Null use case
```

```
connmgr.state == UseCaseSelectionCompleted
```

6.1.1.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.4 Roles

6.1.1.4.1 ConnectionManager

Signature

```
role ConnectionManager{
    const setof{UseCase}     useCases;
    const setof{Transition} allowedTransitions;
    UseCase                  currentUseCase;
    UseCaseSelectionState    state;
}
```

Qualifiers

— root

Description

An instance of the role `ConnectionManager` provides functionality to select a use case out of a set of use cases. At any moment in time at most one use case is selected. The platform instance determines the set of use cases from which a use case can be chosen.

Independent Attributes

Table 20

Name	Description
`useCases`	The set of use cases supported by the platform instance. The value of `useCases` is constant because it is set at instantiation time and does not change dynamically. For every platform instance this set contains the `Null use case`.
`allowedTransitions`	This set contains all transitions that are allowed by the platform instance. In many cases this set will contain all possible transitions implying that there are no restrictions. The value of `allowedTransitions` is constant because it is set at instantiation time and does not change dynamically.
`currentUseCase`	The use case that is currently selected. This attribute is only valid when the state is equal to `UseCaseSelectionCompleted`.
`state`	Indicates whether a use case selection is in progress or completed.

Invariants

— (`state == UseCaseSelectionCompleted`) **==>** `currentUseCase` is valid

— `currentUseCase` in `useCases`

— `Null use case` in `useCases`

— For all `transition` in `allowedTransitions`

  — `transition.to` in `useCases`

  — `transition.from` in `useCases`

Instantiation

See subclause 6.1.1.3.3.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Streaming Behavior
The `ConnectionManager` itself does not expose streaming behavior. However, each use case corresponds with a configuration of the component instances that are part of the platform instance. The streaming behavior of each use case: number and types of inputs, number and types of outputs, connections within the use case etc. will have to be specified by the platform instance documentation.

Active Behavior
The `ConnectionManager` has standard notification behavior (see `uhIConnMgrPlfSelUseCaseNtf`).

6.1.1.4.2 ConnectionManagerClient

Signature
role `ConnectionManagerClient {}`

Qualifiers
— actor

Description
A `ConnectionManagerClient` represents a user of the interface `uhIConnMgrPlfSelUseCase`.

6.1.1.5 Interfaces

6.1.1.5.1 uhIConnMgrPlfSelUseCase

Qualifiers
None.

Description
This interface is meant for selecting the current use case and for retrieval of interfaces of component instances that are part of that use case.

This interface is specific for each platform instance (the platform name Plf is part of the interface name). The specification described in this document serves as template for each platform instance specific instantiation.

Interface ID
`Defined as part of the platform instance documentation.`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIConnMgrPlfSelUseCaseNtf   *pINotify,
    [in] UInt32                        cookie,
    [in] uhConnMgrPlf_Notifications_t  notifs);
```

Qualifiers

— subscribe-function

6.1.1.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIConnMgrPlfSelUseCaseNtf   *pINotify,
    [in] UInt32                        cookie,
    [in] uhConnMgrPlf_Notifications_t  notifs);
```

Qualifiers

— unsubscribe-function

6.1.1.5.1.3 SelectUseCaseX

Signature

uhErrorCode_t Select*UseCaseX* (...);

Qualifiers

— asynchronous

— single-threaded

Description

This function serves as a generic example. A platform instance will have a separate function for each use case it supports (see subclause 6.1.1.1.4).

This function performs the switch from the current use case to `UseCaseX`, and makes `UseCaseX` the current use case afterwards. The actual completion of the change is signaled through the `OnUseCaseSelected` notification.

The switch will be as seamless as possible. If needed and possible the system will mute (using the system mute) the output signals to prevent visual and audible artifacts.

Component instances that are part of the current use case, but not part of `UseCaseX` will be destroyed. Component instances that are not part of the current use case, but are part of `UseCaseX` will be created. Component instances that are part of both the current as well as `UseCaseX` will not be stopped and keep their state, unless specified otherwise in the platform instance documentation.

The client has the obligation to release all interfaces to component instances that are not part of `UseCaseX` before calling this function. During the transition the client can use the interfaces from the components that are present in both the old and the new use case.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 21

Name	Description
...	The parameters representing the data associated with the use case.

Return Values
Standard.

Precondition
— `connmgr.state == UseCaseSelectionCompleted`

— `(connmgr.currentUseCase,UseCaseX)` in `connmgr.allowedTransitions`

— The client has released all interfaces of components that are not part of `UseCaseX`.

Action
— Modify `connmgr.state`

Postcondition
`connmgr.state == UseCaseSelectionInProgress`

Asynchronous Action
— For all `comp` in `connmgr.currentUseCase.compSet` and not(`comp` in `UseCaseX.compSet`):

  — destroy `comp`

— For all `comp` in `UseCaseX.compSet` and `not(comp` in `connmgr.currentUseCase.compSet)`:

  — create `comp`

— `connmgr.currentUseCase = UseCaseX`

— `connmgr.state = UseCaseSelectionCompleted`

— Notify `OnUseCaseSelected`(*)

Asynchronous Postcondition
— `connmgr.state == UseCaseSelectionCompleted`

6.1.1.5.1.4 GetInstance

Signature

```
uhErrorCode_t GetInstance (
    [in]  uhConnMgrPlf_CompId_t   compId,
    [in]  uhRefIid_t              riid,
    [out,iid_is(riid)] pVoid     *ppIntf);
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description

Tests whether the root role of the specified component instance provides the requested interface. If this is the case, a pointer to an interface with the specified IID is returned. If the component instance does not provide the requested interface, an error is returned and `*ppIntf` is set to Null.

If the `GetInstance` function returns a valid interface pointer (i.e. not a null pointer) the reference counter of the interface will have been incremented. The client has the obligation to release the interface using `Release` (see Annex B of ISO/IEC 23004-1) when it no longer needs the interface.

Parameters

Table 22

Name	Description
`compId`	Identifies a component instance of the current use case.
`riid`	Pointer to the IID of the interface (pointer) to be returned.
`ppIntf`	Output parameter returning either a pointer to an interface with IID equal to *riid or Null.

Return Values

Table 23

Name	Description
`uh_OK`	Indicates that the component instance provides the requested interface.
`uh_ERR_PLF_NO_INTERFACE`	Indicates that the component instance does not provide the requested interface.

Precondition

— `connmgr.state == UseCaseSelectionCompleted`

— A `comp` exists in `connmgr.currentUseCase.compSet` such that `comp.cid == compId`.

Case 1

— `*riid` in `comp.iidSet`

Action

None.

Postcondition

— `*ppIntf == &intf`, where `intf` is of type `*riid` and provided by `comp`

— `result == uh_OK`

Case 2

— not(`*riid in comp.iidSet`)

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— `*ppIntf == Null`

— `result == uh_ERR_PLF_NO_INTERFACE`

6.1.1.5.1.5 GetContainedComponents

Signature
```
uhErrorCode_t GetContainedComponents(
    [in]  uhConnMgrPlf_UseCaseId_t  useCaseId,
    [out] pUInt32                   pComps0to31,
    [out] pUInt32                   pComps32to63,
    ...);
```

Qualifiers
— synchronous

— thread-safe

Description
This function can be used to query for the component instances that are part of the specified use case. It can be used by the client to find out for which components it has to release interfaces prior to selecting a new use case and to find out for which components it can request interfaces after this use case has been selected.

Parameters

Table 24

Name	Description
`useCaseId`	Identifies the use case
`pComps0to31`	Pointer to a bit array in which bit i indicates whether the component with uh`ConnMgr_CompId_t` equal to i is contained in the specified use case.
`pComps32to63`	Pointer to a bit array in which bit i indicates whether the component with uh`ConnMgr_CompId_t` equal to (i + 32) is contained in the specified use case.
`...`	Etc.

The number of parameters of this function depends on the number of component instances described in the platform instance. When the total number of component instances is less or equal to 32, one parameter suffices. When the total number of component instances is greater than 32 and less or equal to 64, two parameters are needed, etc.

Return Values
Standard.

Precondition
— `connmgr.state == UseCaseSelectionCompleted`

— A use case `uc` exists in `connmgr.useCases` such that `uc.id == useCaseId`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— For all bits i in `*pComps0_to31`

  — Let `comp` be such that `comp.cid == i`

  — i == 1 <==> `comp` in `uc`

— For all bits i in `*pComps32_to63`

  — Let `comp` be such that `comp.cid == (i + 32)`

  — i == 1 <==> `comp` in `uc`

Etc.

6.1.1.5.2 uhIConnMgrPlfSelUseCaseNtf

Qualifiers
— call-back

Description
This interface is platform specific and should be provided by each client that wants to subscribe to notifications of events that occur when selecting a use case. The `Subscribe` function of `uhIConnMgrPlfSelUseCase` can be used to subscribe to these events.

Interface ID
`Defined as part of the platform instance documentation.`

Remarks
— The notification interface should at least contain the functions specified in this document. For a specific platform instance it may be decided to extend the interface with additional functions.

6.1.1.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                         cookie,
    [in] uhConnMgrPlf_Notifications_t   notifs);
```

Qualifiers
— onsubscriptionchanged-function

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.5.2.2 OnUseCaseSelected

Signature

```
[async] Void OnUseCaseSelected (
    [in] UInt32                    cookie,
    [in] uhConnMgrPlf_UseCaseId_t useCaseId);
```

Qualifiers
None.

Description
This function is called by the Connection Manager when it has completed the selection of a use case (see `SelectUseCaseX`) and also each time a provider of the notification interface (re-)enables subscription to the `OnUseCaseSelected` notification. In the latter case the notification is only sent to the new subscriber and only when the subscription was not already enabled. The initial use-case is returned in case there was no use-case selected at that point.

Upon and after reception of this event, the client can retrieve interfaces to the component instances of the newly selected use case.

Parameters

Table 25

Name	Description
`cookie`	The cookie identifying the subscription to which this notification applies.
`useCaseId`	Identifies the use case that has been selected.

Return Values
None.

Precondition
— `True`

Action
— Any callback-compliant action.

Postcondition
— `True`

6.1.2 uhlFatErr

6.1.2.1 Concepts

6.1.2.1.1 Types of Errors

An “error” is defined as a condition that, if not handled, can cause a system to crash or malfunction. Many errors can be avoided by using appropriate development and verification techniques, but it is a fact that of life

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

that most systems contain errors. When these errors manifest themselves at execution time, they should somehow be “handled” so as to avoid the system to crash or malfunction.

We distinguish three types of errors:

a) Programming errors

These are errors in the design of the software, such as precondition violations, deadlocks, etc. This type of errors is supposed to be detected and handled at development time, e.g. using assert statements while debugging and testing. The platform does not provide special provisions for handling this type of errors. Programming errors that go undetected will often manifest themselves at run-time by a system crash.

b) Functional errors

These are errors that are supposed to be handled by the client. The server component reporting the error is in a normal state and can continue its task as specified. A typical example is an “increase volume” function that returns a “maximum volume reached” error code. Errors of this type are part of the “protocol” associated with an interface and specified as normal return values. Failing to handle such an error in the appropriate way is considered a programming error.

Functional errors are reported via return codes of the interface functions. If no error occurred, UH_OK is returned (0x00000000). In case a functional error has occurred, a value in the range 0x00000001 to 0x7FFFFFFF is returned (the highest bit is used to differentiate between functional and exceptional errors). The lowest 12 bits are defined by M3W and represent the actual error code, ranging from 0x00000001 to 0x00000FFF. The remaining 19 bits can be used in a platform specific way, e.g. for debugging purposes. When no error has occurred, these 19 bits also have to be zero. This makes the check on UH_OK straightforward. In case an error has occurred, bit masking can be used to obtain the actual error code. An example is given below:

```
errorCode =  f(...);
/* In case of UH_OK, all bits must be zero; no masking required for check */
if (UH_OK == errorCode) {
    /* No error has occurred; this is the normal case.  */
    /* The client continues with normal operation.      */
    ...
} else {
    /* An error has occurred. Only the lowest 12 bits are defined within */
    /* M3W; masking is needed                            */
    switch (errorCode & 0x00000FFF) {
        case UH_ERR_A:
            /* handle error A */
               ...
   etc.
```

c) Exceptional errors

These are errors that are detected by a server component but that are explicitly not resolved by that component (e.g. because of too much effort, too high cost, too low benefit). The errors are indicated by error codes in the range 0x80000000 to 0xFFFFFFFF. Fatal errors are one group of exceptional errors. The fundamental difference with a functional error is that the server component is not in a normal state anymore. In order to handle these fatal errors the platform provides special functionality, referred to as the *fatal* error handler. Fatal errors that occur in the platform are reported by the fatal error handler to the client that typically aborts normal program execution. The other group of exceptional errors consists of non-fatal errors. These are also system-wide errors, but the system is able to continue after their occurrence. Non-fatal errors are also reported via the fatal error handler via a separate interface. Whether an exceptional error is fatal or non-fatal is specified by the platform instance.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.2.1.2 Fatal Error Handling

In the streaming part of a platform there is typically one fatal error handler instance. Its primary purpose is to react to fatal errors that may occur in platform code and abort normal program execution in a controlled way. The abortion of normal program execution can both mean that the total system execution is aborted, or a part of it (depending on the system context) [Where this specification refers to abortion, this should be interpreted as such.]. What happens after abortion is not specified in this document. A typical situation is that the system is reset.

The way the fatal error handler works can be illustrated by means of the object configuration in Figure 8. It shows an object representing a piece of client code calling functions in an interface. While executing the platform code associated with these functions, the client thread may run into a fatal error. If so, the client thread will report the error to the fatal error handler in a *synchronous* way, i.e. the thread is blocked while the fatal error handler processes the error. Note that the function used to report the error is internal to the platform and of no concern to the client of the API; the error reporting interface is implementation-dependent and not part of the API.

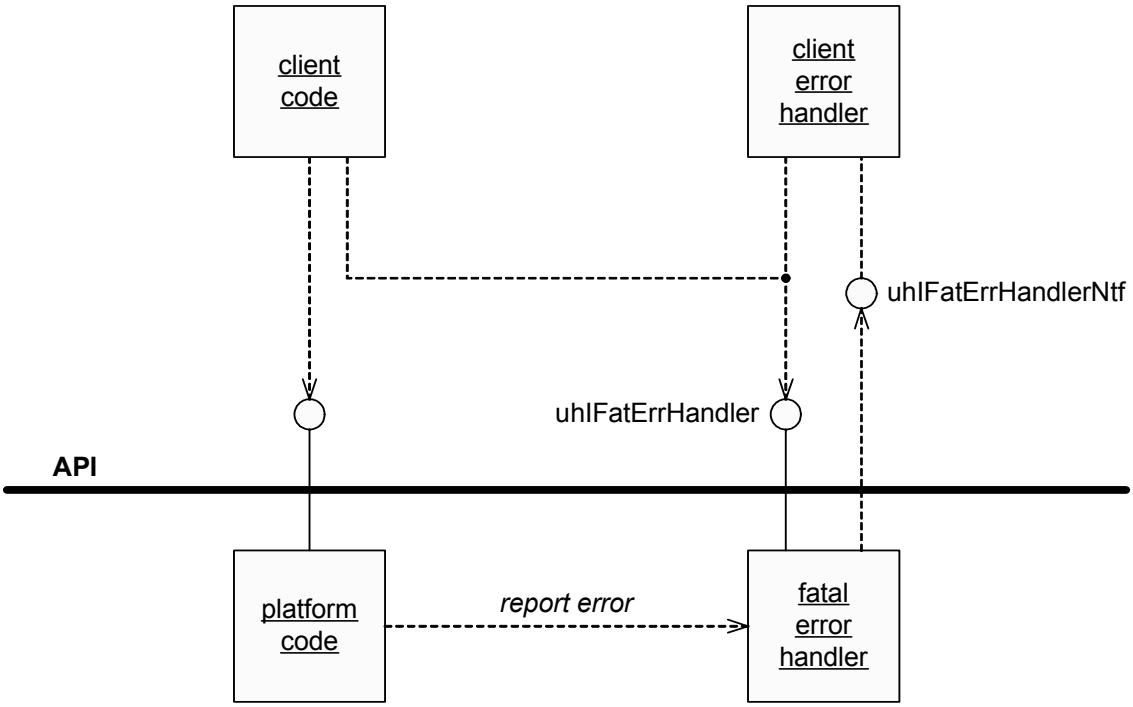


Figure 8 — Typical Fatal Error Handler Configuration

6.1.2.1.3 Client-Specific Fatal Error Handling

In practice, clients may want to handle fatal errors in a specific way. For that purpose the fatal error handler provides a facility to install a client-specific error handler. The standard event subscription mechanism is used for this, i.e. a client error handler can be installed by subscribing to “fatal error” notifications provided by the fatal error handler.

For the purpose of client-specific error handling the notification mechanism is used in a special way. First of all, there may be at most one subscriber to the fatal error notifications, the “client error handler”, which should provide the `uhIFatErrHandlerNtf` notification interface. Having more than one error handler would make no sense. Secondly, the notifications of the fatal error handler are *synchronous* because the fatal error handler should be blocked while the client error handler is handling the error. By installing a client error handler all responsibility for handling the error including abortion is transferred to the client error handler.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

So, when an error is reported to the fatal error handler and assuming a client error handler has been installed, the fatal error handler will notify the client error handler (possibly on the load of the client's thread) by a synchronous call of the OnError function in the `uhIFatErrHandlerNtf` interface of the client error handler:

```
OnError( cookie, error, size, file, line )
```

where `error` is the fatal error code and `file` and `line` are parameters that indicate in which piece of code the error occurred. The `size` parameter is there for marshaling purposes and indicates the size of the `file` string (= length + 1). This notification call does typically not return, since the client error handler typically will abort program execution, usually after making a number of preparations such as logging the error.

If no client error handler is installed, the fatal error handler will itself take care of aborting program execution.

6.1.2.1.4 Fatal Error Handling Protocol

The protocol described above is summarized in the sequence diagrams in Figure 9 and Figure 10, covering the case of an installed client error handler and the case that no client error handler has been installed. Note that in the figures we assumed that the fatal error is reported by a client thread. However, internal platform threads may also report errors.

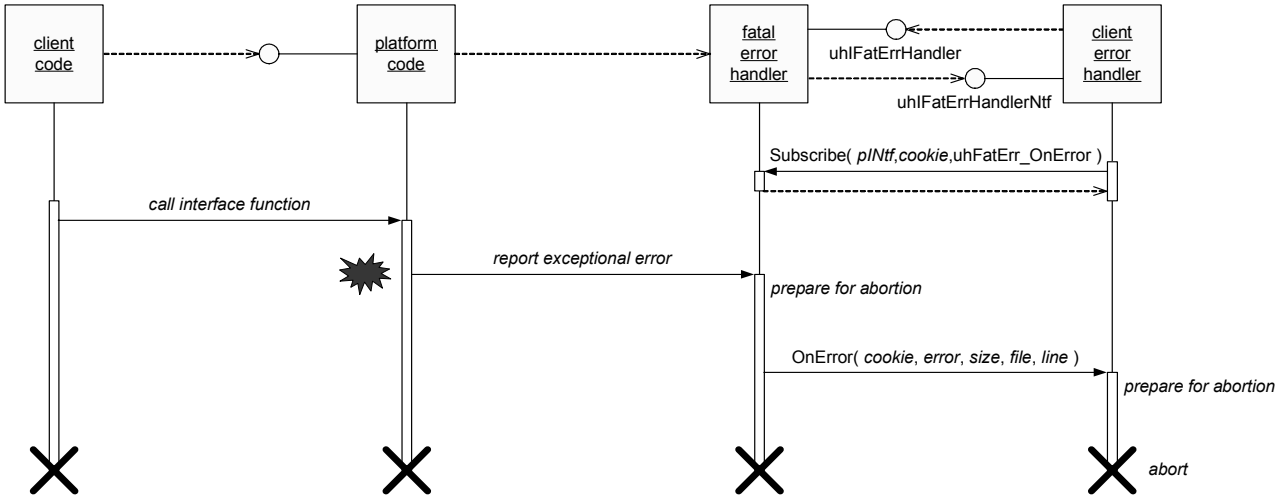


Figure 9 — Fatal Error Handling Protocol with installed Client Error Handler

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

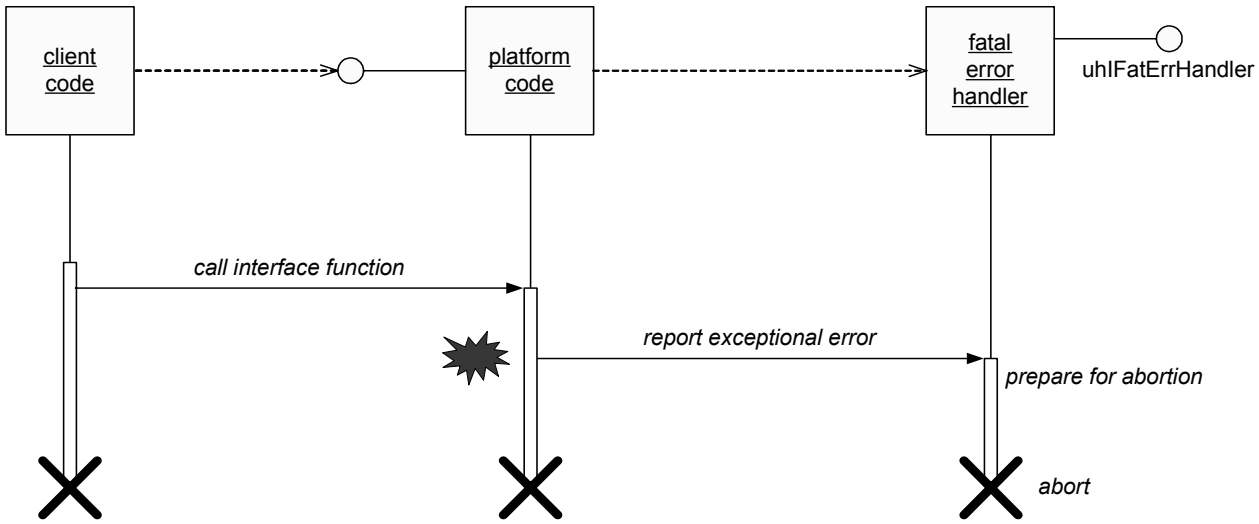


Figure 10 — Fatal Error Handling Protocol without installed Client Error Handler

6.1.2.1.5 Non-fatal Error Handling

Next to the fatal errors, non-fatal errors can also be relevant for a client application. Examples of such errors are 'input sample lost' or 'video frame skipped'. Such errors are non-fatal in the sense that they do not have to lead to aborting program execution. Logging of these errors is useful for several purposes, including:

— testing purposes, e.g. to check certain performance related problems like skipping video frames

— servicing purposes, e.g. to check whether a certain hardware component has intermittent problems

These errors can be logged by the client application in non-volatile memory for later inspection.

Two interfaces `uhIFatErrLog` and `uhIFatErrLogNtf` are used for handling the non-fatal errors. The fatal error handler will deal with these non-fatal errors differently than with the fatal errors. The main difference is that the system continues after a notification call. During execution, the platform code can report non-fatal errors to the fatal error handler via an implementation-dependent interface. The client can subscribe for the notification of non-fatal errors. An occurrence of a non-fatal error will lead to the invocation of the generic error log notification function: `OnError ( cookie, error, nrOfOccurrences, errorData )`

Since non-fatal errors such as 'video frame skipped' can come in a burst mode, a platform instance may decide not to generate a separate notification for the occurrence of each single event. To still be able to inform the client that a certain error has occurred a few times, also the number of occurrences of an error event is identified in a notification call. Furthermore, one additional attribute is provided that can be used for additional error information. Since several errors may be notified within one notification call, it is suggested to use this field to specify the data associated with the last error that occurred.

Which non-fatal errors exist must be defined by the platform instance.

6.1.2.2 Types & Constants

6.1.2.2.1 Public Types & Constants

6.1.2.2.1.1 Error Codes

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.2.2.1.2 uhFatErr_Notification_t

Signature

```
typedef enum _uhFatErr_Notification_t {
    uhFatErr_SubscriptionChanged = 0x00000001,
    uhFatErr_Error               = 0x00000002
} uhFatErr_Notification_t, *puhFatErr_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type are used to represent the individual notification functions in the `uhIFatErrHandlerNtf` interface.

Values

Table 26

Name	Description
`uhFatErr_SubscriptionChanged`	Represents `OnSubscriptionChanged`
`uhFatErr_Error`	Represents `OnError`

6.1.2.2.1.3 uhFatErr_Notifications_t

Signature

```
typedef Int32 uhFatErr_Notifications_t, *puhFatErr_Notifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`FatErr_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints

— None.

6.1.2.2.1.4 uhFatErr_AllNotifications

Signature

```
const uhFatErr_Notifications_t uhFatErr_AllNotifications = 0x00000003;
```

Qualifiers

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set of all values of type uh`FatErr_Notification_t`. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

6.1.2.2.1.5 uhFatErr_LogNtf_t

Signature
```
typedef enum _uhFatErr_LogNtf_t {
    uhFatErr_LogSubscriptionChanged  = 0x00000001,
    uhFatErr_LogError          = 0x00000002
} uhFatErr_LogNtf_t, *puhFatErr_LogNtf_t;
```

Qualifiers
— enum-element

Description
Values of this type are used to represent the individual notification functions in the `uhIFatErrLogNtf` interface.

Values

Table 27

Name	Description
`uhFatErr_LogSubscriptionChanged`	Represents `OnSubscriptionChanged`
`uhFatErr_LogError`	Represents `OnError`

6.1.2.2.1.6 uhFatErr_LogNtfSet_t

Signature
```
typedef Int32 uhFatErr_LogNtfSet_t, *puhFatErr_LogNtfSet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type `uhFatErr_LogNtf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints
— None

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.2.2.1.7 uhFatErr_AllLogNtfs

Signature

```
const uhFatErr_LogNtfSet_t uhFatErr_AllLogNtfs = 0x00000003;
```

Qualifiers

None.

Description

Defines the set of all values of type `uhFatErr_LogNtf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

6.1.2.2.2 Model Types & Constants

None.

6.1.2.3 Logical Component

6.1.2.3.1 Interface-Role Model

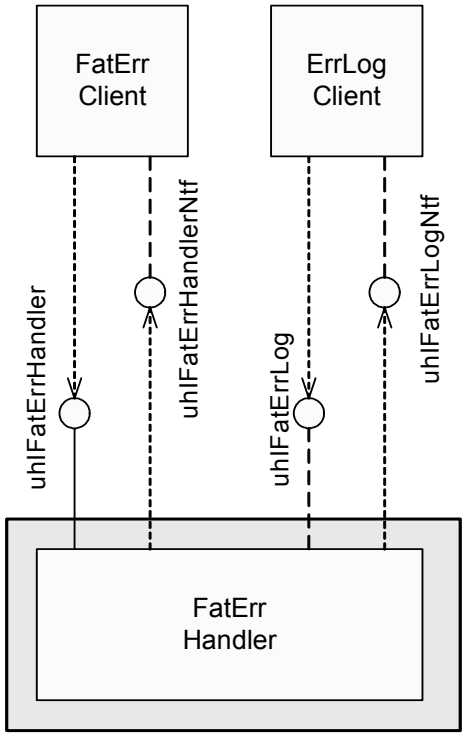


Figure 11 — Interface-Role Model

The role `FatErrHandler` represents (the behavior of) a fatal error handler. In the streaming part of a platform there is typically one fatal error handler instance. The role `FatErrClient` represents a client-specific error handler for handling fatal errors. The role `ErrLogClient` represents a client-specific handler that takes care of logging non-fatal errors. Within a system, there is typically at most one instantiation of each of these two `client` roles. The `uhIFatErrHandler` interface provided by a fatal error handler can be used to install a client-specific error handler using the event-subscription mechanism. A client specific error handler

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

should provide a `uhIFatErrHandlerNtf` interface that is used by the fatal error handler to notify the occurrence of an fatal error. The optionally provided `uhIFatErrLog` interface can be used to install a client-specific error logger using the event-subscription mechanism. A client specific error logger should then provide a `uhIFatErrLogNtf` interface that is used by the fatal error handler to notify the occurrence of non-fatal errors.

6.1.2.3.2 Diversity

6.1.2.3.2.1 Provided Interfaces

Table 28

Role	Interface	Presence
`FatErrHandler`	`uhIFatErrHandler`	mandatory
	`uhIFatErrLog`	optional
`FatErrClient`	`uhIFatErrHandlerNtf`	optional
`ErrLogClient`	`uhIFatErrLogNtf`	optional

6.1.2.3.2.2 Configurable Items

None.

6.1.2.3.2.3 Constraints

None.

6.1.2.3.3 Instantiation

6.1.2.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 29

Type	Object	Multiplicity
`FatErrHandler`	`feh`	1

6.1.2.3.3.2 Initial State

There are no constraints on the initial state of a logical component instance.

6.1.2.3.4 Execution Constraints

No execution constraints apply to the Fatal Error Handling logical component; the interfaces `uhIFatErrHandler` and `uhIFatErrLog` are thread-safe.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.2.4 Roles

6.1.2.4.1 FatErrHandler

Signature
role `FatErrHandler {}`

Qualifiers
— root

Description
An instance of role `FatErrHandler`, called a *fatal error handler*, handles fatal errors that may occur in the platform by aborting program execution and provides facilities to install a client error handler using the event subscription mechanism. If a client error handler is installed, the fatal error handler delegates the handling of a fatal error to the client error handler.

Next to the fatal errors, also non-fatal errors can be handled by a *fatal error handler*. These errors can be reported to a client error logger via the event subscription mechanism.

Independent Attributes
None.

Dependent Attributes
None.

Instantiation
See subclause 6.1.2.3.3.

Streaming Behavior
None.

Active Behavior
The `FatErrHandler` has standard notification behavior (see `uhIFatErrHandlerNtf` and `uhIFatErrLogNtf`) except for the event related to the `uhIFatErrHandlerNtf` interface described below:

Table 30

Event	Action

Some thread in the system reports that a fatal error with error code *c* occurred at line number *n* in the file with name *f* and transfers control to the fatal error handler.	The fatal error handler performs the following action
	If(a client error handler has been installed)
	— Notify OnError(*,*c*,f,n) in a *synchronous* way.
	— If (client did not abort)
	— Prepare for aborting program execution.
	— Abort.
	Else
	— Prepare for aborting program execution.
	— Abort.

Execution Constraints
— Fatal errors are processed in a sequential way (implying that only the first reported error will be handled, due to aborting of program execution).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks

— Note that the `OnError` of interface `uhIFatErrHandlerNtf` notification is performed as a *synchronous* function call, implying that the thread reporting the fatal error remains pending.

— What happens after the abortion of program execution is deliberately left unspecified.

— In this API specification we do not specify how exceptional errors are reported within the platform to a fatal error handler. This is an implementation issue that is not observable by the client of the API. The same holds for the non-fatal errors.

6.1.2.4.2 FatErrClient

Signature

```
role FatErrClient {}
```

Qualifiers

— actor

Description

An instance of role `FatErrClient`, called a *client error handler*, represents a custom error handler that may be installed by an application as a “delegate” of the fatal error handler. A client error handler is installed by subscribing to the `OnError` notifications of the fatal error handler on interface `uhIFatErrHandlerNtf`.

Remarks

— The execution constraints associated with the fatal error handler imply that no more than one client error handler can be installed.

— Note that a call to the client error handler to report a fatal error should does typically not return; see subclause 6.1.2.4.5.2. If this condition is not met and the function returns, the fatal error handler will abort program execution.

6.1.2.4.3 ErrLogClient

Signature

```
role ErrLogClient {}
```

Qualifiers

— actor

Description

An instance of role `ErrLogClient`, called a *client error logger*, represents a custom error logger that may be installed by an application as a “delegate” of the fatal error handler for non-fatal errors. A client error logger is installed by subscribing to the `OnError` notifications of the fatal error handler on interface `uhIFatErrLogNtf`.

6.1.2.4.4 InterfacesuhIFatErrHandler

Qualifiers

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This is the control interface of the fatal error handler allowing a client to subscribe and unsubscribe to error notifications.

Interface ID
`uuid( 5854CE21-5427-4d97-B477-BCBE7DF695E6 )`

Execution Constraints
— At any given moment the number of subscribers to the `OnError` event may be at most one.

6.1.2.4.4.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIFatErrHandlerNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhFatErr_Notifications_t notifs );
```

Qualifiers
— subscribe-function

6.1.2.4.4.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIFatErrHandlerNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhFatErr_Notifications_t notifs );
```

Qualifiers
— unsubscribe-function

6.1.2.4.5 uhIFatErrHandlerNtf

Qualifiers
— callback

Description
This interface should be provided by each client error handler so as to receive notifications of the occurrence of fatal errors.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
uuid(D7F64618-555E-4896-86C3-E0AE047C6174)

6.1.2.4.5.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhFatErr_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

6.1.2.4.5.2 OnError

Signature
```
Void OnError (
    [in] UInt32 cookie,
    [in] uhErrorCode_t error,
    [in] UInt32 size,
    [in, size_is(size)] String file,
    [in] UInt32 line );
```

Qualifiers
None.

Description
This function is used by a fatal error handler to notify a client error handler that a fatal error occurred and that it should abort execution (possibly partially); see the Active Behavior of `FatErrHandler` in subclause 6.1.2.4.1. Before aborting, the client error handler can make any necessary preparations such as logging the error.

Parameters

Table 31

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`error`	The error code of the fatal error whose occurrence is being notified.
`size`	The string size of the file parameter (= string length + 1).
`file`	The name of the file where the fatal error occurred.
`line`	The number of the line where the fatal error occurred.

Return Values
None.

Precondition
```
0x80000000 <= error <= 0xFFFFFFFF
```

— `size = strlen( file ) + 1`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action

— Prepare for the (possibly partial) abortion of program execution.

— Abort (possibly partially).

Postcondition

True

Remarks

— This function (the client error handler) typically never returns, by aborting the system execution, or part of it (depending on the system context). The abort is typically followed by a restart to bring the system in a defined state again. If this condition is not met and the function returns, the fatal error handler will abort the system execution, or part of it, and will typically do a restart after the abort.

— What happens after the (possibly partial) abortion of program execution is deliberately left unspecified.

6.1.2.4.6 uhIFatErrLog

Qualifiers

None.

Description

This is the control interface of the fatal error handler allowing a client to subscribe and unsubscribe to non-fatal error notifications for logging purposes.

Interface ID

uuid(4DA5775C-CC66-487c-8DFA-A9589F3278AB)

Execution Constraints

None

6.1.2.4.6.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIFatErrLogNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhFatErr_LogNtfSet_t notifs );
```

Qualifiers

— subscribe-function

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

6.1.2.4.6.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIFatErrLogNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhFatErr_LogNtfSet_t notifs );
```

Qualifiers

— unsubscribe-function

6.1.2.4.7 uhIFatErrLogNtf

Qualifiers

— callback

Description

This interface should be provided by each client error handler so as to receive notifications of the occurrence of non-fatal errors.

Interface ID

```
uuid( 6F59485E-4C9E-4ffc-8CFD-BC47517DEE1C )
```

6.1.2.4.7.1 OnSubscriptionChanged

Signature

```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhFatErr_LogNtfSet_t notifs );
```

Qualifiers

— onsubscriptionchanged-function

6.1.2.4.7.2 OnError

Signature

```
[async] Void OnError (
    [in] UInt32        cookie,
    [in] uhErrorCode_t error,
    [in] UInt32        nrOfOccurrences,
    [in] Int32         errorData );
```

Qualifiers

None.

Description

This function is called by the fatal error handler to notify the occurrence of a non-fatal error. To avoid that notifications are being made in a very high frequency for errors that occur in a burst mode, the fatal error handler may report multiple occurrences of the same error within one notification.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 32

Name	Description
cookie	The value that was passed by the client at subscription time.
error	The error code of the non-fatal error whose occurrence is being notified
nrOfOccurences	The number of occurrences of the error since the last time this error was notified.
errorData	Specific data related to this error. Since multiple errors may have occurred, it is suggested to record the data of the last error (this is to be determined by the platform instance).

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

Remarks
None.

6.1.3 uhIamAlive

6.1.3.1 Concepts

6.1.3.1.1 The watchdog

The I am Alive logical component (IAA) supports watchdog functionality. The watchdog is an entity that needs to receive a heartbeat from the platform every few seconds (see Figure 12). If this heartbeat does not come (in time), it initiates a reboot of the platform. The heartbeat is an “I am alive” notification provided by the IAA. The IAA periodically checks if the entire platform is still alive and if this is the case, it issues an “I am alive” notification.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

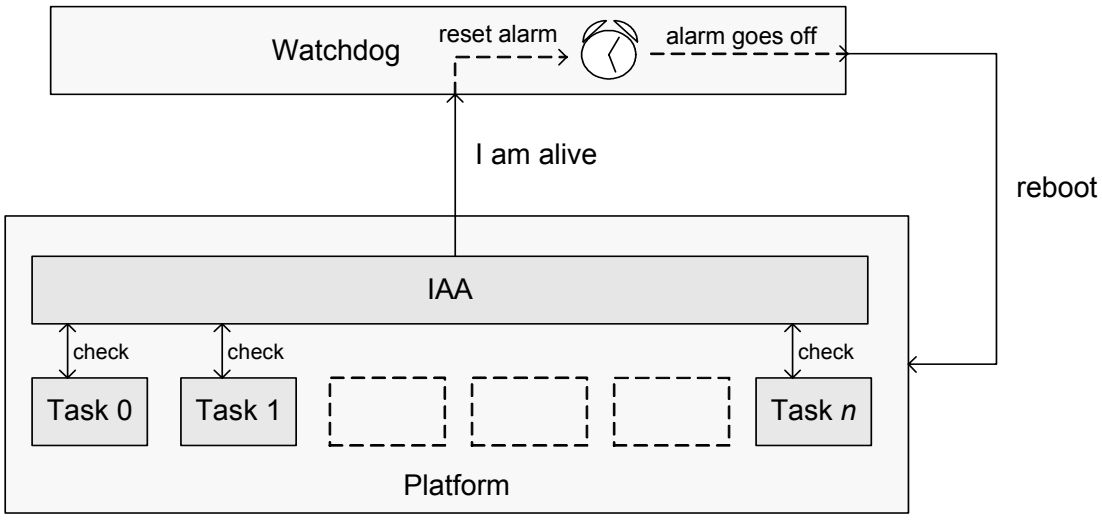


Figure 12 — Interface-Role Model of the IAA.

The IAA (grey box) consists of the `IamAliveNotifier` role. This role provides an interface (`uhIIamAlive`) for monitoring the state (alive/in deadlock) of the platform. The control interface is meant for use by the client (`IamAliveNotifierClient`). This client can optionally provide a call-back interface (`uhIIamAliveNtf`) allowing it to get notified of events in the `IAA`.

6.1.3.1.2 Diversity

6.1.3.1.2.1 Provided Interfaces

Table 33

Role	Interface	Presence
`IamAliveNotifier`	`uhIIamAlive`	mandatory
`IamAliveNotifierClient`	`uhIIamAliveNtf`	optional

6.1.3.1.2.2 Configurable Items

Table 34

Role	Attribute
`IamAliveNotifier`	`ntfInterval`

6.1.3.1.2.3 Constraints

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.3.1.3 Instantiation

6.1.3.1.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 35

Type	Object	Multiplicity
`IamAliveNotifier`	`iaan`	1

6.1.3.1.3.2 Initial State

No constraints on the initial state.

6.1.3.1.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

6.1.3.2 Roles

6.1.3.2.1 IamAliveNotifier

Signature

```
role  IamAliveNotifier {
    Uint32 ntfInterval;
}
```

Qualifiers

— root

Description

The `IamAliveNotifier` role provides means to check if the platform is alive.

Independent Attributes

Table 36

Name	Description
`ntfInterval`	The maximum time (in milliseconds) in between two consecutive “I am alive” notifications.

Dependent Attributes

None.

Invariants

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Instantiation
See subclause 6.1.3.1.3.

Streaming Behavior
None.

Active Behavior
The `IamAliveNotifier` has standard notification behavior (see `uhIIamAliveNtf`).

6.1.3.2.2 IamAliveNotifierClient

Signature
role `IamAliveNotifierClient {}`

Qualifiers
— actor

Description
A `IamAliveNotifierClient` represents a user of the `uhIIamAlive` interface.

6.1.3.3 Interfaces

6.1.3.3.1 uhIIamAlive

Qualifiers
None.

Description
This interface allows the client to subscribe to the “I am alive” events issued by the `IamAliveNotifier`. He can also query the notification interval used by the `IamAliveNotifier`.

Interface ID
`uuid( 60B6D267-1EB4-44e5-909D-B570E9EFCE27 )`

Remarks
In the following function specifications, the name `iaan` will be used to refer to the provider of this interface; see subclause 6.1.3.1.3.1.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.3.3.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIIamAliveNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhIamAlive_Notifications_t notifs );
```

Qualifiers

— subscribe-function

6.1.3.3.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIIamAliveNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhIamAlive_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

6.1.3.3.1.3 GetNotificationInterval

Signature

```
uhErrorCode_t GetNotificationInterval ( [out] pUInt32 pInterval );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the notification interval of the `IamAliveNotifier`.

Parameters

Table 37

Name	Description
`pInterval`	Output parameter returning the notification interval (in milliseconds) of the `IamAliveNotifier`.

Return Values

Standard.

Precondition

`True`

Action

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition

— `*pInterval == iaan.ntfInterval`

6.1.3.3.2 uhIIamAliveNtf

Qualifiers

— callback

Description

This is the notification interface to be provided by clients of the IAA logical component that wish to subscribe to notifications of events of the IAA logical component.

Interface ID

`uuid( 9A528730-25E7-4528-A052-8AFB6E0CA5A1 )`

6.1.3.3.2.1 OnSubscriptionChanged

Signature

```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhIamAlive_Notifications_t notifs );
```

Qualifiers

— onsubscriptionchanged-function

6.1.3.3.2.2 OnIamAlive

Signature

```
[async] Void OnIamAlive ( [in] UInt32 cookie );
```

Qualifiers

None.

Description

This function is called by the `IamAliveNotifier` on a periodic basis (the time in between two consecutive notifications is at most `ntfInterval`) to notify the client that the platform is alive.

Parameters

Table 38

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values

None.

Precondition

`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action

Postcondition
True

6.1.4 uhIOutConf

6.1.4.1 Concepts

6.1.4.1.1 Introduction

The input of an Output Configuration component is an uncompressed audio signal with a given channel configuration. The Output Configuration component processes these channels to get to an audio output signal with a channel configuration that corresponds to wishes of the client(s).

The so-called processes (i.e. algorithms) that can be applied to the audio input signal have been grouped into so-called Standards according to the (company) providers of the algorithm:

— A basic group gathers the non-proprietary processes;

— A Dolby group gathers the Dolby processes;

— etc.

A process can have limitations on the channel configuration of the audio input signal to which it can be applied: for instance, the Dolby Pro-Logic algorithm can only be applied to a stereo input signal. For a same process applied, the channel configuration of the output signal can also differ between input signals with different channel configuration: for instance, when the SRS Wow process is applied to a mono input signal, the output signal is a mono signal, and when it is applied to a stereo signal, the output signal is a stereo signal.

Consequently, for each possible channel configuration of the audio input signal it can be necessary to apply a different process to fit as much as possible with requirements towards the channel configuration of the audio output signal. For instance, when the audio playback capability of the system does not fit with the audio material - for example, a television with only left and right speakers fed with a multi-channel audio stream - an appropriate process should be selected to get an output signal with a channel configuration fitting this limitation.

For each channel configuration of the audio input signal (i.e. 1/0, 2/0, 2/1, 2/2, etc.), a number of processes are supported. The application of these processes on input streams with a specific input channel configuration can result in different channel configurations of the output signal. Therefore per input channel configuration, one can establish a list of output channel configurations that can be obtained, by application of one of the processes supported for this input channel configuration.

In other words, only a subset of the supported Standards and within these Standards only a subset of the supported processes will perform a specific input channel configuration to output channel configuration "transformation" (i.e combination).

For each of the possible input channel configuration, a preference can be set towards one of the possible output channel configurations and towards the process that should be used (along with the Standard the process is belonging to) in order to obtain an output signal with this channel configuration.

In case the channel configuration of the input signal changes, the Output Configuration can thus autonomously and transparently select which of the Standards it should use and which of the processes it should apply.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This is depicted in red in the example of Figure 13 in the case of a change in the input stream to a stereo signal. The Basic Standard should be used (since this is the Standard selected by the client for such channel configuration of the input signal) with Front 3 Stereo as the process applied (since this is the process selected by the client when the audio input signal is stereo).

Whenever another change of the channel configuration of the input stream occurs towards a configuration with three front channels and two surround (a.k.a. rear) channels, this will imply - in the example depicted in Figure 13 - the use of the Dolby Standard and the aplication of the Dolby Virtual process to the audio input signal - as depicted in light purple in Figure 13.

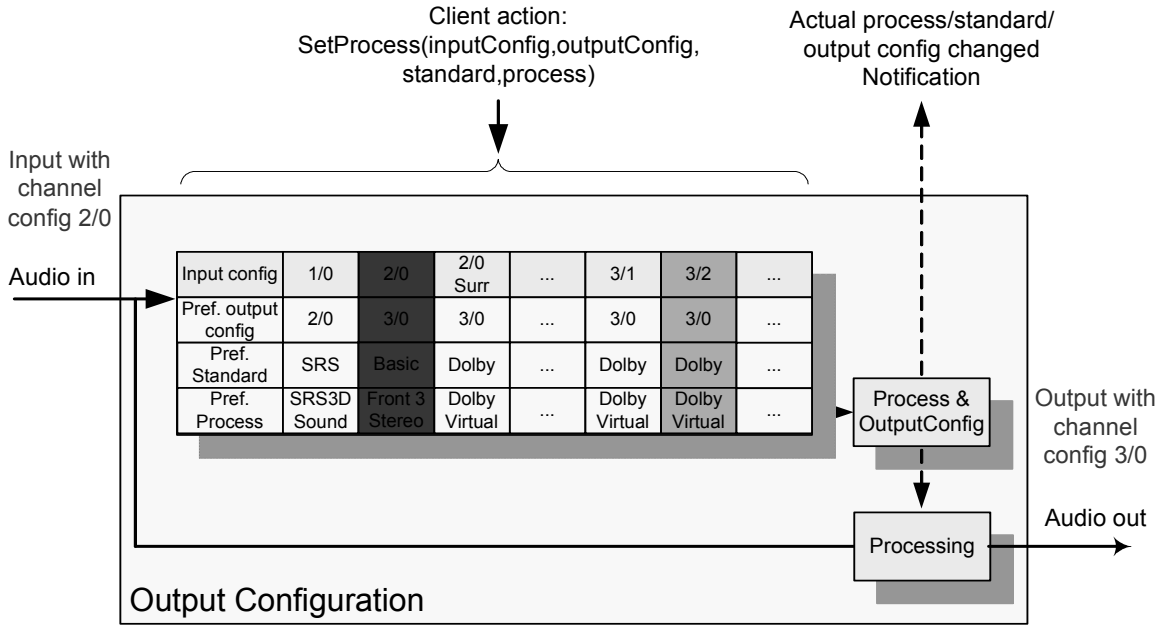


Figure 13 — Example of the Output Configuration Mechanism

For some of the processes the amount of effect that should be used when the process is applied is configurable. Control of the desired amount of effect to be used is provided to the client(s) in the same function used to set the process to be applied for a specific input channel configuration.

6.1.4.2 Types & Constants

6.1.4.2.1 Public Types & Constants

6.1.4.2.1.1 Error Codes

None.

6.1.4.2.1.2 uhOutConf_ValStat_t

```
Signature
typedef enum _uhOutConf_ValStat_t {
    uhOutConf_ValStatInvalid      = -2,
    uhOutConf_ValStatUnknown      = -1,
    uhOutConf_ValStatKnown        =  1
} uhOutConf_ValStat_t, *puhOutConf_ValStat_t;
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 39

Name	Description
uhOutConf_ValStatInvalid	The property/properties are temporarily not available.
uhOutConf_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhOutConf_ValStatKnown	The property/properties are available and determined.

6.1.4.2.1.3 uhOutConf_Config_t

Signature

```
typedef enum _uhOutConf_Config_t {
    uhOutConf_ConfigInvalid      = 0x80000000,
    uhOutConf_ConfigUnknown      = 0x40000000,
    uhOutConf_Config1s0          = 0x00000001,
    uhOutConf_Config2s0          = 0x00000002,
    uhOutConf_Config2s0Surround  = 0x00000004,
    uhOutConf_Config2s1          = 0x00000008,
    uhOutConf_Config2s2          = 0x00000010,
    uhOutConf_Config3s0          = 0x00000020,
    uhOutConf_Config3s1          = 0x00000040,
    uhOutConf_Config3s2          = 0x00000080,
    uhOutConf_Config3s3          = 0x00000100,
    uhOutConf_Config3s4          = 0x00000200,
    uhOutConf_Config4s2          = 0x00000400,
    uhOutConf_Config4s4          = 0x00000800,
    uhOutConf_Config5s2          = 0x00001000,
    uhOutConf_Config5s3          = 0x00002000
} uhOutConf_Config_t, *puhOutConf_Config_t;
```

Qualifiers
— enum-element

Description
Values of this type represent channel configurations of an audio stream.

Values

Table 40

Name	Description
uhOutConf_ConfigInvalid	The channel configuration is temporarily not available.
uhOutConf_ConfigUnknown	The channel configuration is available, yet it could not be determined.
uhOutConf_Config1s0	Mono signal: only a single channel is present in the audio stream.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_Config2s0	Stereo signal: left and right channels. The audio channels do not contain Dolby Surround encoded signal.
uhOutConf_Config2s0Surround	Stereo signal: left and right channels. The audio stream contains a (Dolby, DTS, etc) Surround encoded stereo signal.
uhOutConf_Config2s1	Left, right, and mono surround channels.
uhOutConf_Config2s2	Left, right, surround left, and surround right channels.
uhOutConf_Config3s0	Left, center, and right channels.
uhOutConf_Config3s1	Left, center, right, and mono surround channels.
uhOutConf_Config3s2	Left, center, right, surround left, and surround right channels.
uhOutConf_Config3s3	Left, center, right, rear left, rear center, and rear right channels.
uhOutConf_Config3s4	Left, center, right, left surround, left-back surround, right-back surround, and right surround channels.
uhOutConf_Config4s2	Left, left-center, right-center, right, surround left, and surround right channels.
uhOutConf_Config4s4	Left, left-center, right-center, right, surround left, surround left-center, surround right-center, and surround right channels.
uhOutConf_Config5s2	Left, left-center, center, right-center, right, surround left, and surround right channels.
uhOutConf_Config5s3	Left, left-center, center, right-center, right, surround left, surround center, and surround right channels.

6.1.4.2.1.4 uhOutConf_ConfigSet_t

Signature

```
typedef Int32 uhOutConf_ConfigSet_t, *puhOutConf_ConfigSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of channel configurations.

Constraints

None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.2.1.5 uhOutConf_Standard_t

Signature

```
typedef enum _uhOutConf_Standard_t {
    uhOutConf_StdNone    = 0x00000001,
    uhOutConf_StdBasic   = 0x00000002,
    uhOutConf_StdDolby   = 0x00000004,
    uhOutConf_StdIncred  = 0x00000008,
    uhOutConf_StdSrs     = 0x00000010,
    uhOutConf_StdDts     = 0x00000020
} uhOutConf_Standard_t, *puhOutConf_Standard_t;
```

Qualifiers

— enum-element

Description

Values of this type represent Standards to be applied to audio input signals.

Values

Table 41

Name	Description
uhOutConf_StdNone	There is no process to apply to the audio input stream. The audio input stream is outputted without any processing.
uhOutConf_StdBasic	The process to be applied to the audio input signal is among a list of basic processes.
uhOutConf_StdDolby	The process to be applied to the audio input signal is among a list of Dolby processes.
uhOutConf_StdIncred	The process to be applied to the audio input signal is among a list of Incredible processes.
uhOutConf_StdSrs	The process to be applied to the audio input signal is among a list of SRS processes.
uhOutConf_StdDts	The process to be applied to the audio input signal is among a list of DTS processes.

6.1.4.2.1.6 uhOutConf_StandardSet_t

Signature

```
typedef Int32
uhOutConf_StandardSet_t, *p
uhOutConf_StandardSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of standards.

Constraints

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.2.1.7 uhOutConf_Process_t

Signature
```
typedef UInt32 uhOutConf_Process_t, *puhOutConf_Process_t;
```

Qualifiers
None.

Description
Defines the "union" of the Standards as a subrange of the integers. That is, a value of this type is a value of type `uhOutConf_BasicProcess_t`, `uhOutConf_DolbyProcess_t`,

uhOutConf_IncredProcess_t,

uhOutConf_SrsProcess_t, or

uhOutConf_DtsProcess_t or is equal to `uhOutConf_ProcessNone`. In order to interpret a value of this type correctly the Standard should be indicated. For example, the value `0x00000004` can be interpreted in one of five ways:

- as the value `uhOutConf_BasicPseudoStereo` of Standard `uhOutConf_StdBasic`;
- as the value `uhOutConf_DolbyProLogic` of Standard `uhOutConf_StdDolby`;
- as the value `uhOutConf_IncredSurround` of Standard `uhOutConf_StdIncred`;
- as the value `uhOutConf_Srs3dSound` of Standard `uhOutConf_StdSrs`;
- as the value `uhOutConf_DtsSurroundEs` of Standard `uhOutConf_StdDts`.

Likewise, the value `0x80000000` can only be interpreted as the value `uhOutConf_ProcessNone` of Standard `uhOutConf_StdNone`.

Constraints
- Each value of type `uhOutConf_Process_t` is a valid value of at least one of the following types: `uhOutConf_BasicProcess_t`, `uhOutConf_DolbyProcess_t`,
- `uhOutConf_IncredProcess_t`,
- `uhOutConf_SrsProcess_t`, or
- `uhOutConf_DtsProcess_t` or is equal to `uhOutConf_ProcessNone`.

Remarks
A value of this type is normally used in combination with a value of type `uhOutConf_Standard_t` to indicate the Standard to which the value belongs.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.2.1.8 uhOutConf_ProcessSet_t

Signature

```
typedef Int32 uhOutConf_ProcessSet_t, *puhOutConf_ProcessSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of values of type `uhOutConf_Process_t`.

Constraints

None.

6.1.4.2.1.9 uhOutConf_Ntf_t

Signature

```
typedef enum _uhOutConf_Ntf_t {
    uhOutConf_SubscriptionChanged  = 0x00000001,
    uhOutConf_ActualProcessChanged = 0x00000002
} uhOutConf_Ntf_t, *puhOutConf_Ntf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhIAdrcNtf.

Values

Table 42

Name	Description
`uhOutConf_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhOutConf_ActualProcessChanged`	Represents `OnActEnabledChanged`.

6.1.4.2.1.10 uhOutConf_NtfSet_t

Signature

```
typedef Int32 uhOutConf_NtfSet_t, *puhOutConf_NtfSet_t;
```

Qualifiers

— enum-set

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type `uhOutConf_Ntf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints
None.

6.1.4.2.1.11 uhOutConf_AllNtfs

Signature
```
const uhOutConf_NtfSet_t uhOutConf_AllNtfs = 0x00000003;
```

Qualifiers
None.

Description
Defines the set of all values of type `uhOutConf_Ntf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

6.1.4.2.1.12 uhOutConf_BasicProcess_t

Signature
```
typedef enum _uhOutConf_BasicProcess_t {
    uhOutConf_BasicSpatial       = 0x00000001,
    uhOutConf_BasicUpmix1To2     = 0x00000002,
    uhOutConf_BasicPseudoStereo  = 0x00000004,
    uhOutConf_Basic3dSurround    = 0x00000008,
    uhOutConf_BasicHall          = 0x00000010,
    uhOutConf_BasicMatrix        = 0x00000020,
    uhOutConf_BasicFront3Stereo  = 0x00000040,
    uhOutConf_BasicDownmix       = 0x00000080
} uhOutConf_BasicProcess_t, *puhOutConf_BasicProcess_t;
```

Qualifiers
— enum-element

Description
Values of this type represent basic processes that can be applied to audio input signals.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 43

Name	Description
uhOutConf_BasicSpatial	Spatial process. Support: for mono and for stereo input signals. When the input signal is: — A mono signal, this process delivers an output signal with a front left and a front right channel with a pseudo-stereo effect; — A stereo signal, this process widens the sound impression, giving the impression that the front left and front right speakers are at a greater distance from each other. The output signal is a stereo signal.
uhOutConf_BasicUpmix1To2	"Basic" upmix from 1 channel to 2 channels. Support: for mono input signals. The output signal is a signal with two channels: a signal with a front left and a front right channel is delivered after proper scaling (fixed per standards, for energy conservation) of the mono input signal. The signals contained in the front left channel and in the front right channel are identical.
uhOutConf_BasicPseudoStereo	Pseudo Stereo process. Support: for mono input signals. This process delivers an output signal with a front left, a front right, and a rear (surround) channel.
uhOutConf_Basic3dSurround	3D Surround process. Support: for stereo input signals. This process (re-)generates a 3D surround sound from a stereo input signal. This process delivers a stereo signal.
uhOutConf_BasicHall	Hall process. Support: for any input channel configuration (i.e. excluding uhOutConf_ConfigInvalid and uhOutConf_ConfigUnknown). This process delivers an output signal with a front left, a front right, a rear left, and a rear right channel. When the input signal: — Does not contain surround channel(s), this process creates artificial ones (a surround left and a surround right channel); — Is a multi-channel signal containing a front center channel, this front center channel is merged into the front left and the front right channel.
uhOutConf_BasicMatrix	Matrix process. This process can be supported for mono and for stereo input signals. This process provides basic surround decoding facility, delivering a signal with a front left, a front right, a front center, and a single surround channel.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_BasicFront3Stereo	Front 3 Stereo process. Support: for stereo and for multi-channel input signals. This process delivers a signal with a front left, a front right, and a front center channel. When the input signal is: — A plain stereo signal, a center channel separation is performed, providing front left, front right, and front center channels. — A possibly Surround encoded stereo signal: — If supported, Dolby Pro-Logic decoding is first applied and then a downmix of the surround channels is performed to obtain a signal with a front left, a front right, and a front center channel; — If not supported, a center channel separation is performed, providing a front left, a front right, and a front center channel. — A multi-channel signal that does not contain a front center channel, then this process performs a center channel separation, and delivers a signal with a front left, a front right, and a front center channel. — A multi-channel signal containing surround channels, a downmix of these channels to a signal with a front left, a front right, and a front center channel is performed. — A multi-channel signal with a center channel and without surround channels, the input signal is passed on unchanged.
uhOutConf_BasicDownmix	Downmix process. Support: for stereo and for multi-channel input signals, depending on the selected output channel configuration. The channel configuration of the signal that is delivered is compliant to the selected output configuration.

Remarks

— When the selected process is a downmixing process (i.e uhOutConf_BasicDownmix) and when the audio input stream contains a single rear (i.e. surround) channel and the required output configuration does not contain a single rear channel but at least rear left and right channels, then the single rear signal is routed after proper scaling (fixed per standards) to the rear left and right channels.

— With respect to downmixing, the coefficients used to obtain the required output configuration are usually already fixed and specified in the standard with which the signal was potentially encoded. The coefficients can also be embedded in the encoded stream itself as can be the case for an originally DTS encoded stream.

— Table 44 defines, per input channel configuration, for each of the basic processes the output configuration that is delivered, depending on the selected output configuration.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 44 — Channel configuration of the output signal in case a basic process is applied to the input stream.

Process	Input Config / Output Config	1/0	2/0	2/0Surr	2/1	2/2	3/0	3/1	3/2	3/3	3/4	4/2	4/4	5/2	5/3
Spatial	2/0	2/0	2/0	2/0	-	-	-	-	-	-	-	-	-	-	-
Upmix 1 To 2	2/0	2/0	-	-	-	-	-	-	-	-	-	-	-	-	-
Pseudo Stereo	2/1	2/1	-	-	-	-	-	-	-	-	-	-	-	-	-
3D Surround	2/0	-	2/0	2/0	-	-	-	-	-	-	-	-	-	-	-
Hall	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
Matrix	3/1	3/1	3/1	3/1	-	-	-	-	-	-	-	-	-	-	-
Front 3 Stereo	3/0	-	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0
Downmix	1/0	-	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0
	2/0	-	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0
	2/2	-	-	-	2/2	-	-	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
	3/0	-	-	-	-	-	-	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0
	3/2	-	-	-	-	-	-	3/2	-	3/2	3/2	3/2	3/2	3/2	3/2

6.1.4.2.1.13 uhOutConf_DolbyProcess_t

Signature

```
typedef enum _
uhOutConf_DolbyProcess_t {
    uhOutConf_DolbySurroundDownmix    = 0x00000001,
    uhOutConf_Dolby3Stereo            = 0x00000002,
    uhOutConf_DolbyProLogic           = 0x00000004,
    uhOutConf_DolbyProLogic2          = 0x00000008,
    uhOutConf_DolbyProLogic2x         = 0x00000010,
    uhOutConf_DolbyHeadphone          = 0x00000020,
    uhOutConf_DolbyVirtual            = 0x00000040,
    uhOutConf_DolbyVirtualSpeaker     = 0x00000080,
    uhOutConf_DolbyVirtualSpeakerWide = 0x00000100
}

uhOutConf_DolbyProcess_t, *p

uhOutConf_DolbyProcess_t;
```

Qualifiers

— enum-element

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description

Values of this type represent Dolby processes that can be applied to audio input signals.

Values

Table 45

Name	Description
uhOutConf_ DolbySurroundDownmix	Downmix to a Dolby Surround encoded stereo signal. Support: for input signals that have a number of front channels bigger or equal to 2 (excluding uhOutConf_Config2s0 and uhOutConf_Config2s0Surround). The signal that is delivered contains a front left and a front right channel Dolby Surround encoded.
uhOutConf_ Dolby3Stereo	Dolby 3 stereo process. Support: for possibly Surround encoded stereo input signals and for multi-channel input signals. This process provides a front left, a front right, and a front center signal. When the input signal is: — A possibly Surround encoded stereo signal, Dolby Pro-Logic decoding is first applied and then a downmix of the surround channels is performed to obtain a signal with a front left, a front right, and a front center channel. — A multi-channel signal containing surround channel(s), a downmix of these channels into the front left and the front right channel is performed. — A multi-channel signal containing only a front left and a front right channel: no channel center is derived, surround information is downmixed into the front left and front right channels. — A multi-channel signal with a center channel and without surround channels, the input signal is passed on unchanged.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
`uhOutConf_`	Dolby Pro-Logic process.
`DolbyProLogic`	Support: for stereo input signals.
	When the input signal is:
	— A plain stereo signal:
	— If the input signal was previously encoded according to a standard with which the application of the Dolby Pro-Logic process conflicts (for instance Dolby Digital), the input signal is passed on unchanged;
	— If the input signal was not previously encoded or was encoded according to a standard with which the application of the Dolby Pro-Logic process does not conflict, this process delivers, depending on the selected output configuration:
	— Either a signal with a front left, a front right, a front center, and a (single) surround channel;
	— Either a signal with a front left, a front right, and a (single) surround signal (the signal in the front center channel is merged into the signals delivered in the front left and the front right channel).
	— A possibly Surround encoded stereo signal, this process delivers depending on the selected output configuration:
	— Either a signal with a front left, a front right, a front center, and a (single) surround channel;
	— Either a signal with a front left, a front right, and a (single) surround signal (the signal in the front center channel is merged into the signals delivered in the front left and the front right channel).

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_ DolbyProLogic2	Dolby Pro-Logic II process. Support: for stereo input signals. When the input signal is: — A plain stereo signal: — If the input signal was previously encoded according to a standard with which the application of the Dolby Pro-Logic II process conflicts (for instance Dolby Digital), the input signal is passed on unchanged; — If the input signal was not previously encoded or was encoded according to a standard with which the application of the Dolby Pro-Logic II process does not conflict, this process delivers, depending on the selected output configuration: — Either a signal with a front left, a front right, a front center, and full range stereo surround channels; — Either a signal with a front left, a front right, a surround left, and a surround right channel (the signal in the front center channel is merged into the signals delivered in the front left and the front right channel). — A possibly Surround encoded stereo signal, this process delivers depending on the selected output configuration: — Either a signal with a front left, a front right, a front center and full range stereo surround channels; — Either a signal with a front left, a front right, a surround left, and a surround right channel (the signal in the front center channel is merged into the signals delivered in the front left and the front right channel).
uhOutConf_ DolbyProLogic2x	Dolby Pro-Logic IIx process. Support: for stereo input signals and for input signals with three front channels and two surround channels. This process delivers depending on the selected output configuration: — Either a signal with a front left, a front center, a front right, a left surround, a center surround, and a right surround channel; — Either a signal with a front left, a front center, a front right, a left surround, a left back, a right back, and a right surround channel.
uhOutConf_ DolbyHeadphone	Dolby Headphone process. Support: for stereo and multi-channel input signals. This process delivers a stereo signal. When the input signal is: — A stereo signal, this process modifies the input signal into a spacious, natural sound, stereo signal for headphone playback; — A multi-channel signal, this process virtualizes the multi-channel signal into a stereo signal with surround effects over any set of headphones.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_ DolbyVirtual	Dolby Virtual Dolby process. Support: for stereo and multi-channel input signals. This process delivers depending on the selected output configuration: — Either a stereo signal; — Either a signal with a front left, a front right and possibly a front center channel. — When the input signal is — A plain stereo: — If the input signal was previously encoded according to a standard with which the application of the Dolby Pro-Logic process conflicts (for instance Dolby Digital), the input signal is passed on unchanged; — If the input signal was not previously encoded or was encoded according to a standard with which the application of the Dolby Pro-Logic process does not conflict, Dolby Pro-Logic processing is first performed, and then virtualizing is applied. The surround channel is virtualized and routed towards front left and front right channels. Depending on the selected output configuration, the process delivers: — Either a virtual surround stereo signal (the front center channel, resulting from the Dolby Pro-Logic processing, is merged into the signals delivered in the front left and the front right channel); — Either a multi-channel signal with a front left, a front right, and a front center channel. — A possibly Surround encoded stereo signal, Dolby Pro-Logic processing is first performed, and then virtualizing is applied. The surround channel is virtualized and routed towards front left and front right channels. Depending on the selected output configuration, the process delivers: — Either a virtual surround stereo signal (the front center channel, resulting from the Dolby Pro-Logic processing, is merged into the signals delivered in the front left and the front right channel); — Either a multi-channel signal with a front left, a front right, and a front center channel. — A multi-channel signal, the surround channel(s), if present in the input signal, is virtualized and routed towards the front left and the front right channel. Depending on the selected output configuration, this process delivers — Either a stereo signal (the center channel, if present in the input signal, being distributed over the front left and the front right channel); — Either a multi-channel signal with a front left, a front center (if originally present in the audio input stream) and a front right channel.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_ DolbyVirtualSpeaker	Dolby Virtual Speaker process, Reference mode. Support: for stereo and multi-channel input signals. This process delivers a stereo signal. When the input signal is: — A plain stereo signal, this process creates a wider two-channel environment; — A possibly Surround encoded stereo signal: — If supported, Dolby Pro-Logic II processing is first performed and then virtualizing is applied, a virtual surround stereo signal of higher quality is delivered; — Otherwise, this process creates a wider two-channel environment. — A multi-channel signal, this process simulates a multi-channel surround sound listening environment by virtualizing the multi-channel input signal to a stereo signal.
uhOutConf_ DolbyVirtualSpeakerWide	Virtual Speaker process, Wide mode. Support: for stereo and multi-channel input signals. This process delivers a stereo signal. It provides a wider, more spacious front image when the two speakers are close together, with the same virtual surround sound as the Reference mode of the Virtual Speaker process. When the input signal is: — A plain stereo signal, this process creates a wider two-channel environment; — A possibly Surround encoded stereo signal: — If supported, Dolby Pro-Logic II processing is first performed and then virtualizing is applied, a virtual surround stereo signal of higher quality is delivered; — Otherwise, this process creates a wider two-channel environment. — A multi-channel signal, this process simulates a multi-channel surround sound listening environment by virtualizing the multi-channel input signal to a stereo signal.

Remarks

Table 46 defines, per input channel configuration, for each of the Dolby processes the output configuration that is delivered, depending on the selected output configuration.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 46 — Channel configuration of the output signal in case a Dolby process is applied to the input stream.

	Input Config	**1/0**	**2/0**	**2/0 Surr**	**2/1**	**2/2**	**3/0**	**3/1**	**3/2**	**3/3**	**3/4**	**4/2**	**4/4**	**5/2**	**5/3**
Process	**Output Config**														
Dolby Surround Downmix	2/0Surr	-	-	-	2/0 Surr	2/0 Surr	2/0 Surr	2/0 Surr	2/0 Surr	2/0 Surr	2/0 Surr	2/0 Surr	2/0 Surr	2/0 Surr	2/0 Surr
Dolby 3 Stereo	3/0	-	-	3/0	-	-	-	3/0	3/0	3/0	3/0	-	-	3/0	3/0
	2/0	-	-	-	2/0	2/0	-	-	-	-	-	2/0	2/0	-	-
Dolby Pro-Logic	3/1	-	3/1 (AC-3: 2/0)	3/1	-	-	-	-	-	-	-	-	-	-	-
	2/1	-	2/1 (AC-3: 2/0)	2/1	-	-	-	-	-	-	-	-	-	-	-
Dolby Pro-Logic II	3/2	-	3/2 (AC-3: 2/0)	3/2	-	-	-	-	-	-	-	-	-	-	-
	2/2	-	2/2 (AC-3: 2/0)	2/2	-	-	-	-	-	-	-	-	-	-	-
Dolby Pro-Logic IIx	3/4	-	3/4	3/4	-	-	-	-	3/4	-	-	-	-	-	-
	3/3	-	3/3	3/3	-	-	-	-	3/3	-	-	-	-	-	-
Dolby Headphone	2/0	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0
Dolby Virtual	3/0	-	3/0 (AC-3: 2/0)	3/0	-	-	-	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0
	2/0	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0
Dolby Virtual Speaker	2/0	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0
Dolby Virtual Speaker Wide	2/0	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.2.1.14 uhOutConf_IncredProcess_t

Signature

```
typedef enum _
uhOutConf_IncredProcess_t {
    uhOutConf_IncredMono       = 0x00000001,
    uhOutConf_IncredStereo     = 0x00000002,
    uhOutConf_IncredSurround   = 0x00000004,
    uhOutConf_Incred3dSurround = 0x00000008
}
uhOutConf_IncredProcess_t, *p
uhOutConf_IncredProcess_t;
```

Qualifiers

— enum-element

Description

Values of this type represent Incredible processes that can be applied to audio input signals.

Values

Table 47

Name	Description
uhOutConf_IncredMono	Incredible Mono process. Support: for mono input signals. This process delivers a stereo signal, where the left channel is identical to the input signal, while the right channel contains the input signal on which a 180 degrees phase shift at 800 Hz has been performed. This process adds a stereo-like impression on a mono signal.
uhOutConf_IncredStereo	Incredible Stereo process. Support: for stereo input signals. This process delivers a widened stereo signal, immersing the listener into the screen action.
uhOutConf_IncredSurround	Incredible Surround process. Support: for stereo input signals. This process delivers a signal with a front left and a front right channel, and two phantom speakers that enlarge the stereo effect. This process is an enhanced Spatial effect, giving the impression that the speakers are positioned further apart than they actually are.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_Incred3dSurround	Incredible 3D Surround process. Support: for possibly Surround encoded stereo signals and multi-channel input signals. When the input signal is: — A possibly Surround encoded stereo signal, this process enhances the surround aspect. The Lt and Rt channels are converted towards a signal with a front left, a front center, a front right and a (single) surround channel. The (single) surround channel is virtualized and routed to the front left and the front right channel. An output signal with a front left, a front center and a front right channel is delivered. — A multi-channel signal, if existing, the surround channel(s) is virtualized and routed to the front left and the front right channel. In the output signal, the front left, the front center (if originally present in the audio input signal) and the front right channel is active.

Remarks

Table 48 defines, per input channel configuration, for each of the Incredible processes the output configuration that is delivered, depending on the selected output configuration.

Table 48 — Channel configuration of the output signal in case an Incredible process is applied to the input stream.

	Input Config	1/0	2/0	2/0 Surr	2/1	2/2	3/0	3/1	3/2	3/3	3/4	4/2	4/4	5/2	5/3
Process	**Output Config**														
Incredible Mono	2/0	2/0	-	-	-	-	-	-	-	-	-	-	-	-	-
Incredible Stereo	2/0	-	2/0	-	-	-	-	-	-	-	-	-	-	-	-
Incredible Surround	2/0	-	2/0	-	-	-	-	-	-	-	-	-	-	-	-
Incredible 3D Surround	3/0	-	-	3/0	-	-	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0	3/0
	2/0	-	-	-	2/0	2/0	-	-	-	-	-	-	-	-	-

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.2.1.15 uhOutConf_SrsProcess_t

Signature

```
typedef enum _
uhOutConf_SrsProcess_t {
    uhOutConf_Srs                   = 0x00000001,
    uhOutConf_Srs3dMono             = 0x00000002,
    uhOutConf_Srs3dSound            = 0x00000004,
    uhOutConf_SrsHeadphone          = 0x00000008,
    uhOutConf_SrsWow                = 0x00000010,
    uhOutConf_SrsTruSurround        = 0x00000020,
    uhOutConf_SrsTruSurroundXt      = 0x00000040,
    uhOutConf_SrsTruSurroundXtHeadph = 0x00000080,
    uhOutConf_SrsCircleSurround     = 0x00000100,
    uhOutConf_SrsCircleSurround2    = 0x00000200
}
uhOutConf_SrsProcess_t, *p
uhOutConf_SrsProcess_t;
```

Qualifiers

— enum-element

Description

Values of this type represent SRS processes that can be applied to audio input signals.

Values

Table 49

Name	Description
uhOutConf_	SRS process.
Srs	Support: for stereo input signals. This process delivers a stereo signal closer to the original intention of the artist and to a live performance. This process restores the spatial information that is usually masked by traditional recording and playback techniques.
uhOutConf_	SRS 3D Mono process.
Srs3dMono	Support: for mono input signal. This process delivers a mono or stereo signal: the channel configuration of the signal that is delivered is compliant to the selected output configuration. This process creates a 3D stereo effect on a mono signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_	SRS 3D Sound process.
Srs3dSound	Support: for mono and stereo input signals. This process delivers a stereo signal. When the input signal is: — A mono signal, it is first converted into a stereo signal, by using constant phase filters. Then the stereo signal is broadened and expanded, resulting in a three-dimensional sound field that fills the listening environment. — A stereo signal, this process delivers a broadened and expanded stereo signal, resulting in a three-dimensional sound field that fills the listening environment.
uhOutConf_	SRS Headphone process.
SrsHeadphone	Support: for stereo and for multi-channel input signals. This process delivers a stereo signal where the sound is repositioned from inside the listener's head to outside the listener's ears, creating a more natural and immersive listening experience.
uhOutConf_	SRS Wow process.
SrsWow	Support: for mono and stereo signals. This process produces a taller and wider soundstage with enhanced bass performance. When the input signal is: — A mono signal, the output signal is a mono signal; — A stereo signal, the output signal is a stereo signal.
uhOutConf_	SRS TruSurround process.
SrsTruSurround	Support: for stereo and for multi-channel input signals. This process delivers a stereo signal. This process delivers a full and immersive virtual surround sound experience on stereo channels.
uhOutConf_	SRS TruSurround XT process.
SrsTruSurroundXt	Support: for stereo and for multi-channel input signals. This process delivers a stereo signal. This process is based on the TruSurround technology and includes SRS Dialog clarity for improvement of dialog intelligibility, to create a 3D sound image.
uhOutConf_	SRS TruSurround XT Headphone process.
SrsTruSurroundXtHeadph	Support: for stereo and for multi-channel input signals. This technology delivers a stereo signal. This process provides a surround sound listening experience over headphones.
uhOutConf_	SRS Circle Surround process.
SrsCircleSurround	Support: stereo input signal. This process delivers signal with a front left, a front right, a front center, a surround left, and a surround right channel. True stereo is provided on the surround channels.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_ SrsCircleSurround2	SRS Circle Surround II process. Support: for mono and stereo input signals. This process delivers depending on the selected output channel configuration: — Either a signal with a front left, a front right, a front center, and stereo surround channels; — Either a signal with a front left, a front right, a front center, a surround left, a surround center, and a right surround channel.

Remarks

Table 50 defines, per input channel configuration, for each of the SRS processes the output configuration that is delivered, depending on the selected output configuration.

Table 50 — Channel configuration of the output signal in case a SRS process is applied to the input stream.

Process	Input Config / Output Config	1/0	2/0	2/0 Surr	2/1	2/2	3/0	3/1	3/2	3/3	3/4	4/2	4/4	5/2	5/3
SRS	2/0	-	2/0	-	-	-	-	-	-	-	-	-	-	-	-
SRS 3D Mono	2/0	-	2/0	-	-	-	-	-	-	-	-	-	-	-	-
	1/0	-	1/0	-	-	-	-	-	-	-	-	-	-	-	-
SRS 3D Sound	2/0	2/0	2/0	-	-	-	-	-	-	-	-	-	-	-	-
SRS Headphone	2/0	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0
SRS Wow	2/0	-	2/0	-	-	-	-	-	-	-	-	-	-	-	-
	1/0	1/0	-	-	-	-	-	-	-	-	-	-	-	-	-
SRS Tru Surround	2/0	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0
SRS Tru Surround XT	2/0	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0
SRS Tru Surround XT Headphone	2/0	-	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0
SRS Circle Surround	3/2	-	3/2	3/2	-	-	-	-	-	-	-	-	-	-	-
SRS Circle Surround II	3/3	3/3	3/3	3/3	-	-	-	-	-	-	-	-	-	-	-
	3/2	3/2	3/2	3/2	-	-	-	-	-	-	-	-	-	-	-

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.2.1.16 uhOutConf_DtsProcess_t

Signature

```
typedef enum _
uhOutConf_DtsProcess_t {
    uhOutConf_DtsNeo6         = 0x00000001,
    uhOutConf_DtsNeo6Music    = 0x00000002,
    uhOutConf_DtsSurroundEs   = 0x00000004
}
uhOutConf_DtsProcess_t, *p
uhOutConf_DtsProcess_t;
```

Qualifiers

— enum-element

Description

Values of this type represent DTS processes that can be applied to audio input signals.

Values

Table 51

Name	Description
uhOutConf_DtsNeo6	DTS NEO:6 process. Support: for stereo input signals (plain and Surround encoded) and for input signals with three front channels and two surround channels. This process delivers depending on the selected output configuration: — Either a signal with a front left, a front right, a front center, and stereo surround channels; — Either a signal with a front left, a front right, a front center, a rear center and stereo surround channels.
uhOutConf_DtsNeo6Music	DTS NEO:6 process, Music mode. Support: for plain stereo input signals. With this mode the subtlety and integrity of the original stereo recording is not diminished as it can be with the usual uhOutConf_DtsNeo6 process. This process delivers depending on the selected output configuration: — Either a signal with a front left, a front right, a front center, and stereo surround channels; — Either a signal with a front left, a front right, a front center, a rear center and stereo surround channels.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhOutConf_DtsSurroundEs	DTS Surround ES process. Support for input signals with three front channels and two surround channels. This process delivers depending on the selected output configuration: — Either a signal with a front left, a front right, a front center, and stereo surround channels; — Either a signal with a front left, a front right, a front center, a rear center and stereo surround channels.

Remarks
Table 52 defines, per input channel configuration, for each of the DTS processes the output configuration that is delivered, depending on the selected output configuration.

Table 52 — Channel configuration of the output signal in case a DTS process is applied to the input stream.

	Input Config	1/0	2/0	2/0 Surr	2/1	2/2	3/0	3/1	3/2	3/3	3/4	4/2	4/4	5/2	5/3
Process	**Output Config**														
DTS Neo:6	3/3	-	3/3	3/3	-	-	-	-	3/3	-	-	-	-	-	-
	3/2	-	3/2	3/2	-	-	-	-	3/2	-	-	-	-	-	-
DTS Neo:6 Music Mode	3/3	-	3/3	3/3	-	-	-	-	3/3	-	-	-	-	-	-
	3/2	-	3/2	3/2	-	-	-	-	3/2	-	-	-	-	-	-
DTS Surround Es	3/3	-	-	-	-	-	-	-	3/3	-	-	-	-	-	-
	3/2	-	-	-	-	-	-	-	3/2	-	-	-	-	-	-

6.1.4.2.1.17 uhOutConf_ProcessNone

Signature
```
const UInt32 uhOutConf_ProcessNone = 0x80000000;
```

Qualifiers
None.

Description
Defines that no processing will be applied to the audio input signal. The input signal is passed on unchanged.

Remarks
This value is only to be used in combination with the value `uhOutConf_StdNone_t` of the type `uhOutConf_Standard_t` to indicate that no processing is to be applied.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.2.2 Model Types & Constants

6.1.4.2.2.1 InOutProcesses

Signature

```
typedef struct _InOutProcesses{
    uhOutConf_ProcessSet_t basicProcesses;
    uhOutConf_ProcessSet_t dolbyProcesses;
    uhOutConf_ProcessSet_t incredProcesses;
    uhOutConf_ProcessSet_t srsProcesses;
    uhOutConf_ProcessSet_t dtsProcesses;
} InOutProcesses, *pInOutProcesses;
```

Qualifiers
None.

Description
Defines for a specific input channel configuration the set of processes that are supported, for each existing Standard, to obtain a specific output channel configuration.

Fields

Table 53

Name	Description
basicProcesses	The set of basic processes that are supported for a specific input channel configuration to obtain a specific output channel configuration.
dolbyProcesses	The set of Dolby processes that are supported for a specific input channel configuration to obtain a specific output channel configuration.
incredProcesses	The set of Incredible processes that are supported for a specific input channel configuration to obtain a specific output channel configuration.
srsProcesses	The set of SRS processes that are supported for a specific input channel configuration to obtain a specific output channel configuration.
dtsProcesses	The set of DTS processes that are supported for a specific input channel configuration to obtain a specific output channel configuration.

Constraints
None.

6.1.4.2.2.2 InputProcesses

Signature

```
typedef struct _InputProcesses{
    UInt32                      nrOutputConfigs;
    uhOutConf_Config_t          pOutputConfigs;
    p
uhOutConf_StandardSet_t   pStandards;
    pInOutProcesses             pProcesses;
} InputProcesses, *pInputProcesses;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines a structure to be used for each supported channel configuration of audio input signals: it contains the output configurations that are supported and that can be obtained, as well as the sets of processes and the associated Standards that can be applied in order to obtain each of these output configurations.

Fields

Table 54

Name	Description
nrOutputConfigs	The number of different output configurations that can be obtained from audio input signals with a specific input channel configuration.
pOutputConfigs	Array containing the output configurations that are supported and that can be obtained from audio input signals with a specific input channel configuration.
pStandards	Array containing for each output configuration that can be obtained, the set of Standards for which at least one process can be used to obtain this output configuration.
pProcesses	Array containing for each output configuration that can be obtained, the sets of processes that can be used in order to obtain this output configuration.

Constraints
None.

6.1.4.2.2.3 EffectRange

Signature
```
typedef struct {
    UInt32 min;
    UInt32 max;
} EffectRange;
```

Qualifiers
None.

Description
Values of this type represent the possible range for the effect amount that can be used when a specific process is applied.

Fields

Table 55

Name	Description
min	The minimum effect amount that can be used when a specific process is applied.
max	The maximum effect amount that can be used when a specific process is applied.

Constraints
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.3 Logical Component

6.1.4.3.1 Interface-Role Model

Figure 14 depicts the interface-role model of the Output Configuration Logical Component. It shows the interfaces and the roles involved with this component.

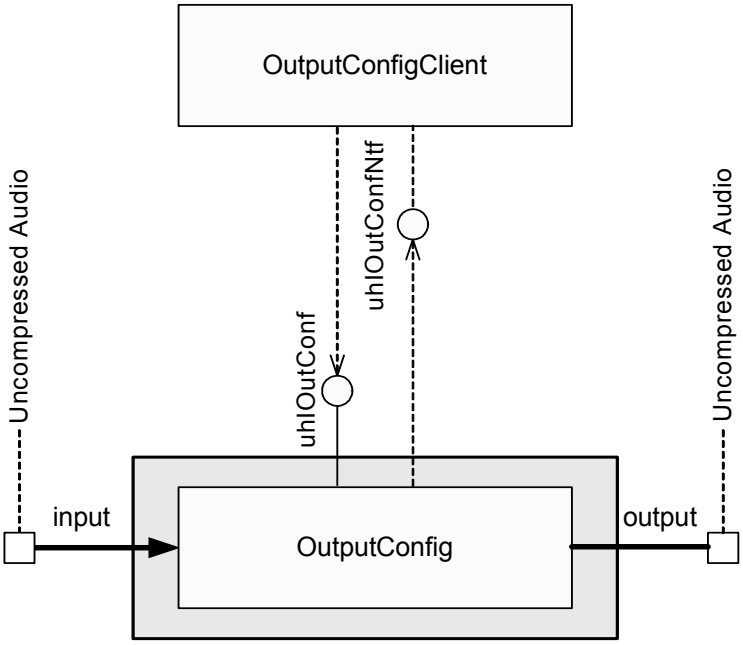


Figure 14 — Interface Role Model

The grey box in Figure 14 denotes an Output Configuration logical component containing a single `OutputConfig` role.

This role provides one mandatory (solid line) control interface (`uhIOutConf`).

The `uhIOutConf`interface provides functionality to control the Standard and the process to be used for each of the supported channel configurations of the audio input stream, as well as functionality to select the channel configuration to be obtained for the audio output stream.

A client can optionally provide a call-back interface, `uhIOutConfNtf`. This allows the client to receive notifications of events in the `OutputConfig`.

6.1.4.3.2 Diversity

6.1.4.3.2.1 Provided Interfaces

Table 56

Role	Interface	Presence
`OutputConfig`	`uhIOutConf`	mandatory
`OutputConfigClient`	`uhIOutConfNtf`	optional

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.3.2.2 Configurable Items

Table 57

Role	Attribute
OutputConfig	nrSuppInputConfigs
	suppInputConfigs
	inputProcesses
	basicEffectRange
	dolbyEffectRange
	incredEffectRange
	srsEffectRange
	dtsEffectRange

6.1.4.3.2.3 Constraints

- nrSuppInputConfigs ≥ 1
- For each i in {0 .. nrSuppInputConfigs-1}
  - !(suppInputConfigs[i] & uhOutConf_ConfigInvalid)
  - !(suppInputConfigs[i] & uhOutConf_ConfigUnknown)
  - inputProcesses[i].nrOutputConfigs ≥ 1
  - suppInputConfigs[i] in inputProcesses[i].pOutputConfigs[]
  - For each j in {0 .. inputProcesses[i].nrOutputConfigs-1}
    - !(inputProcesses[i].pOutputConfigs[j] & uhOutConf_ConfigInvalid)
    - !(inputProcesses[i].pOutputConfigs[j] & uhOutConf_ConfigUnknown)
    - (inputProcesses[i].pStandards[j] & uhOutConf_StdNone) == uhOutConf_StdNone
    - If (inputProcesses[i].pStandards[j] & uhOutConf_StdBasic) == uhOutConf_StdBasic
      - inputProcesses[i].pProcesses[j].basicProcesses != uhOutConf_ProcessNone
    - else
      - inputProcesses[i].pProcesses[j].basicProcesses == uhOutConf_ProcessNone
    - If (inputProcesses[i].pStandards[j] & uhOutConf_StdDolby) == uhOutConf_StdDolby
      - inputProcesses[i].pProcesses[j].dolbyProcesses != uhOutConf_ProcessNone
    - else
      - inputProcesses[i].pProcesses[j].dolbyProcesses == uhOutConf_ProcessNone
    - If (inputProcesses[i].pStandards[j] & uhOutConf_StdIncred) == uhOutConf_StdIncred
      - inputProcesses[i].pProcesses[j].incredProcesses != uhOutConf_ProcessNone
    - else
      - inputProcesses[i].pProcesses[j].incredProcesses == uhOutConf_ProcessNone
    - If (inputProcesses[i].pStandards[j] & uhOutConf_StdSrs) == uhOutConf_StdSrs
      - inputProcesses[i].pProcesses[j].srsProcesses != uhOutConf_ProcessNone

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- else
  - inputProcesses[i].pProcesses[j].srsProcesses == uhOutConf_ProcessNone
- If (inputProcesses[i].pStandards[j] & uhOutConf_StdDts) == uhOutConf_StdDts
  - inputProcesses[i].pProcesses[j].dtsProcesses != uhOutConf_ProcessNone
- else
  - inputProcesses[i].pProcesses[j].dtsProcesses == uhOutConf_ProcessNone

- For each p in uhOutConf_BasicProcess_t
  - basicEffectRange[p].min ≤ basicEffectRange[p].max
- For each p in uhOutConf_DolbyProcess_t
  - dolbyEffectRange[p].min ≤ dolbyEffectRange[p].max
- For each p in
- uhOutConf_IncredProcess_t
  - incredEffectRange[p].min ≤ incredEffectRange[p].max
- For each p in
- uhOutConf_SrsProcess_t
  - srsEffectRange[p].min ≤ srsEffectRange[p].max
- For each p in
- uhOutConf_DtsProcess_t
  - dtsEffectRange[p].min ≤ dtsEffectRange[p].max

6.1.4.3.3 Instantiation

6.1.4.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 58

Type	Object	Multiplicity
OutputConfig	outconf	1

6.1.4.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

- For each j in {0 .. outconf.nrSuppInputConfigs-1}
  - outconf.prefOutputConfig[j] == outconf.suppInputConfigs[j]
  - outconf.prefStandard[j] == uhOutConf_StdNone
  - outconf.prefProcess[j] == uhOutConf_ProcessNone
  - outconf.prefEffect[j] == 0
- outconf.currentInputConfig == uhOutConf_ConfigInvalid
- outconf.currentOutputConfig == uhOutConf_ConfigInvalid
- outconf.currentStatus == uhOutConf_ValStatInvalid
- outconf.currentStandard == uhOutConf_StdNone
- outconf.currentProcess == uhOutConf_ProcessNone
- outconf.currentEffect == 0

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

6.1.4.4 Roles

6.1.4.4.1 OutputConfig

Signature

```
role OutputConfig {
    const UInt32                    nrSuppInputConfigs;
    const uhOutConf_Config_t        suppInputConfigs[nrSuppInputConfigs];
    const InputProcesses            inputProcesses[nrSuppInputConfigs];
    const EffectRange               basicEffectRange[uhOutConf_BasicProcess_t];
    const EffectRange               dolbyEffectRange[uhOutConf_DolbyProcess_t];
    const EffectRange               incredEffectRange[
uhOutConf_IncredProcess_t];
    const EffectRange               srsEffectRange[
uhOutConf_SrsProcess_t];
    const EffectRange               dtsEffectRange[
uhOutConf_DtsProcess_t];
    uhOutConf_Config_t              prefOutputConfig[nrSuppInputConfigs];
    uhOutConf_Standard_t            prefStandard[nrSuppInputConfigs];
    uhOutConf_Process_t             prefProcess[nrSuppInputConfigs];
    UInt32                          prefEffect[nrSuppInputConfigs];
    uhOutConf_Config_t              currentInputConfig;
    uhOutConf_Config_t              currentOutputConfig;
    uhOutConf_ValStat_t             currentStatus;
    uhOutConf_Standard_t            currentStandard;
    uhOutConf_Process_t             currentProcess;
    UInt32                          currentEffect;
}
```

Qualifiers

— root

Description

An `OutputConfig` configures the front and rear channels present in its output stream. An `OutputConfig` provides functionality to the client(s) to select the Standard to be used and the process to apply to the audio input signal for each channel configuration of its audio input stream, as well as functionality to select the channel configuration to be obtained for the output stream.

An `OutputConfig` provides notifications of changes in the channel configuration of its audio output stream, as well as notifications of changes in the actual process applied to the audio input stream, and in the actual Standard associated with the process applied.

Independent Attributes

Table 59

Name	Description
`nrSuppInputConfigs`	The number of input channel configurations that are supported by the `OutputConfig`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
suppInputConfigs	Array containing the input channel configurations supported by the OutputConfig.
inputProcesses	Array containing for each supported input channel configuration the set of output configurations that can be obtained with audio input signals with this channel configuration, the sets of processes and associated Standards that can be used to obtain output streams with these channel configurations. The elements at index x of this array correspond to the input channel configuration in the array suppInputConfigs at index x.
basicEffectRange	Array containing for each basic process the minimum and the maximum effect amount that can be applied. At the minimum effect amount the processing may still be active, the signal is not necessarily passed on unchanged.
dolbyEffectRange	Array containing for each Dolby process the minimum and the maximum effect amount that can be applied. At the minimum effect amount the processing may still be active, the signal is not necessarily passed on unchanged.
incredEffectRange	Array containing for each Incredible process the minimum and the maximum effect amount that can be applied. At the minimum effect amount the processing may still be active, the signal is not necessarily passed on unchanged.
srsEffectRange	Array containing for each SRS process the minimum and the maximum effect amount that can be applied. At the minimum effect amount the processing may still be active, the signal is not necessarily passed on unchanged.
dtsEffectRange	Array containing for each DTS process the minimum and the maximum effect amount that can be applied. At the minimum effect amount the processing may still be active, the signal is not necessarily passed on unchanged.
prefOutputConfig	Array containing for each supported input channel configuration the preferred channel configuration of the output stream of the OutputConfig. An element at index x of this array is the preferred output channel configuration for the input channel configuration in the array suppInputConfigs at index x.
prefStandard	Array containing for each supported input channel configuration the Standard that should be used whenever an audio signal with this channel configuration is present. An element at index x of this array is the preferred Standard for the input channel configuration in the array suppInputConfigs at index x.
prefProcess	Array containing for each supported input channel configuration the process that should be applied whenever an audio signal with this channel configuration is present. An element at index x of this array is the preferred Process for the input channel configuration in the array suppInputConfigs at index x.
prefEffect	Array containing for each supported input channel configuration the effect amount of the preferred process that should be applied whenever an audio signal with this channel configuration is present. An element at index x of this array is the preferred effect amount for the input channel configuration in the array suppInputConfigs at index x.
currentInputConfig	The current channel configuration of the audio input stream.
currentOutputConfig	The current channel configuration of the audio output stream.
currentStatus	Validity state of the information over the current Standard used (currentStandard), over the current process applied (currentProcess) and over the amount of effect used (currentEffect).
currentStandard	The Standard to which the current process currentProcess belongs.
currentProcess	The current process applied to the audio input stream.
currentEffect	The current amount of effect used for the current process currentProcess.

Dependent Attributes
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Invariants

— currentInputConfig in {suppInputConfigs[] | uhOutConf_ConfigInvalid | uhOutConf_ConfigUnknown}
— If currentInputConfig == uhOutConf_ConfigInvalid
  — currentOutputConfig == uhOutConf_ConfigInvalid
  — currentStatus == uhOutConf_ValStatInvalid
— If currentInputConfig == uhOutConf_ConfigUnknown
  — currentOutputConfig == uhOutConf_ConfigUnknown
  — currentStatus == uhOutConf_ValStatUnknown
— For each i in {0 .. nrSuppInputConfigs-1}
  — prefOutputConfig[i] in inputProcesses[i].pOutputConfigs[]
  — Let j be the index of prefOutputConfig[i] in inputProcesses[i].pOutputConfigs[]
  — (prefStandard[i] & inputProcesses[i].pStandards[j]) == prefStandard[i]
  — If (prefStandard[i] == uhOutConf_StdBasic)
    — (prefProcess[i] & inputProcesses[i].pProcesses[j].basicProcesses) == prefProcess[i]
    — basicEffectRange[prefProcess[i]].min ≤ prefEffect[i] ≤ basicEffectRange[prefProcess[i]].max
  else if (prefStandard[i] == uhOutConf_StdDolby)
    — (prefProcess[i] & inputProcesses[i].pProcesses[j].dolbyProcesses) == prefProcess[i]
    — dolbyEffectRange[prefProcess[i]].min ≤ prefEffect[i] ≤ dolbyEffectRange[prefProcess[i]].max
  else if (prefStandard[i] == uhOutConf_StdIncred)
    — (prefProcess[i] & inputProcesses[i].pProcesses[j].incredProcesses) == prefProcess[i]
    — incredEffectRange[prefProcess[i]].min ≤ prefEffect[i] ≤ incredEffectRange[prefProcess[i]].max
  else if (prefStandard[i] == uhOutConf_StdSrs)
    — (prefProcess[i] & inputProcesses[i].pProcesses[j].srsProcesses) == prefProcess[i]
    — srsEffectRange[prefProcess[i]].min ≤ prefEffect[i] ≤ srsEffectRange[prefProcess[i]].max
  else if (prefStandard[i] == uhOutConf_StdDts)
    — (prefProcess[i] & inputProcesses[i].pProcesses[j].dtsProcesses) == prefProcess[i]
    — dtsEffectRange[prefProcess[i]].min ≤ prefEffect[i] ≤ dtsEffectRange[prefProcess[i]].max
  else if (prefStandard[i] == uhOutConf_StdNone)
    — prefProcess[i] == uhOutConf_ProcessNone
    — prefEffect[i] == 0

Instantiation

An OutputConfig is always instantiated as part of an Output Configuration; see subclause 6.1.4.3.3.

Streaming Behavior

The OutputConfig processes the audio input stream according to currentProcess which is the process set in prefProcess[currentInputConfig], this process belongs to the currentStandard Standard, which is the Standard set in prefStandard[currentInputConfig], with currentInputConfig being the current channel configuration of the audio input signal.

After processing, the OutputConfig streams out the audio signal. The channel configuration of the audio output stream is according to the channel configuration in currentOutputConfig, which is identical to the channel configuration set in prefOutputConfig[currentInputConfig] (except in some few cases, e.g.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

when applying some of the Dolby processed on an input signal that was previously encoded according to the AC-3 standard; see Table 46 for details).

Active Behavior
The `OutputConfig` has standard notification behavior (see `uhIOutConfNtf`).

Execution Constraints
None.

6.1.4.4.2 OutputConfigClient

Signature
role `OutputConfigClient {}`

Qualifiers
— actor

Description
An Output Configuration Client represents a user of the `uhIOutConf` interface.

Execution Constraints
None.

6.1.4.5 Interfaces

6.1.4.5.1 uhIOutConf

Qualifiers
None.

Description
This interface of an `OutputConfig` provides functionality to configure the process and the associated Standard that should be used for each supported channel configuration of the audio input stream, as well as functionality to configure the output configuration that should be obtained, functionality to query the current channel configuration of the input and output streams and the current process in use. This interface also provides a subscription and unsubscription function on notifications of changes in the channel configuration of the output stream and changes in the process and associated Standard used.

Interface ID
`uuid( 7C97D8F9-4060-4864-93A7-6CFF73C3F968 )`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIOutConfNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhOutConf_NtfSet_t notifs );
```

Qualifiers

— subscribe-function

6.1.4.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIOutConfNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhOutConf_NtfSet_t notifs );
```

Qualifiers

— unsubscribe-function

6.1.4.5.1.3 GetSuppInputConfigs

Signature

```
uhErrorCode_t GetSuppInputConfigs (
    [out] puhOutConf_ConfigSet_t pInputConfigs );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the set of input channel configurations that are supported by the `OutputConfig`.

Parameters

Table 60

Name	Description
`pInputConfigs`	Output parameter returning the set of input channel configurations that are supported by the `OutputConfig`.

Return Values

Standard.

Precondition

`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
`inputConfigs = 0`

For each `i` in `{0 .. outconf.nrSuppInputConfigs -1}`

```
inputConfigs = ( inputConfigs | outconf.suppInputConfigs[i] )
```

```
*pInputConfigs == inputConfigs
```

6.1.4.5.1.4 GetSuppOutputConfigs

Signature

```
uhErrorCode_t GetSuppOutputConfigs (
    [in] uhOutConf_Config_t inputConfig,
    [out] puhOutConf_ConfigSet_t pOutputConfigs );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns for the specified input channel configuration the set of output channel configurations that are supported and that can be obtained with an audio input stream with this channel configuration.

Parameters

Table 61

Name	Description
`inputConfig`	The input channel configuration for which the set of output channel configurations that are supported and that can be obtained is requested.
`pOutputConfigs`	Output parameter returning for the input channel configuration `inputConfig` the set of output channel configurations that are supported and that can be obtained.

Return Values
Standard

Precondition
`inputConfig in outconf.suppInputConfigs[]`

`Let i be the index of inputConfig in outconf.suppInputConfigs[]`

Action
None.

Postcondition
— `outputConfigs = 0`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For each `j` in `{0 .. outconf.inputProcesses[i].nrOutputConfigs -1}`

— `outputConfigs = ( outputConfigs | outconf.inputProcesses[i].pOutputConfigs[j] ) *pOutputConfigs == outputConfigs`

6.1.4.5.1.5 GetSuppStandards

Signature

```
uhErrorCode_t GetSuppStandards (
    [in] uhOutConf_Config_t inputConfig,
    [in] uhOutConf_Config_t outputConfig,
    [out] p
uhOutConf_StandardSet_t pStandards );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the set of Standards that are supported by the `OutputConfig` for the specified input channel configuration and that can be used with an audio input stream with this input channel configuration to obtain an audio output stream with the specified output channel configuration.

Parameters

Table 62

Name	Description
`inputConfig`	The input channel configuration for which the set of Standards that are supported by the `OutputConfig`, and that can be used to obtain an audio output stream with the output channel configuration `outputConfig`, is requested.
`outputConfig`	The output channel configuration that should be obtained with audio input streams with the input channel configuration `inputConfig` and for which the set of Standards that are supported by the `OutputConfig` and that can be used is requested.
`pStandards`	Output parameter returning the set of Standards that are supported by the `OutputConfig` for the input channel configuration `inputConfig` and that can be used with an audio input stream with this input channel configuration to obtain an audio output stream with the output channel configuration `outputConfig`.

Return Values

Standard

Precondition

— `inputConfig in outconf.suppInputConfigs[]`

— Let `i` be the index of `inputConfig` in `outconf.suppInputConfigs[]`

— `outputConfig` in `outconf.inputProcesses[i].pOutputConfigs[]`

— Let `j` be the index of `outputConfig` in `outconf.inputProcesses[i].pOutputConfigs[]`

Action

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pStandards == outconf.inputProcesses[i].pStandards[j]`

6.1.4.5.1.6 GetSuppProcesses

Signature
```
uhErrorCode_t GetSuppProcesses (
    [in] uhOutConf_Config_t inputConfig,
    [in] uhOutConf_Config_t outputConfig,
    [in] uhOutConf_Standard_t standard,
    [out] puhOutConf_ProcessSet_t pProcesses );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the set of processes of the specifed Standard that are supported by the `OutputConfig` for the specified input channel configuration and that can be used with an audio input stream with this input channel configuration to obtain an audio output stream with the specified output channel configuration.

Parameters

Table 63

Name	Description
`inputConfig`	The input channel configuration for which the set of processes, of the Standard `standard`, that are supported by the `OutputConfig` and that can be used to obtain an audio output stream with the output channel configuration `outputConfig`, is requested.
`outputConfig`	The output channel configuration that should be obtained with audio input streams with the input channel configuration `inputConfig` and for which the set of processes, of the Standard `standard`, that are supported by the `OutputConfig` and that can be used, is requested.
`standard`	The Standard for which the set of processes that are supported by the `OutputConfig` for the input channel configuration `inputConfig` and that can be used with an audio input stream with this input channel configuration to obtain an audio output stream with the output channel configuration `outputConfig`, is requested.
`pProcesses`	Output parameter returning the set of processes, of the Standard `standard`, that are supported by the `OutputConfig` for the input channel configuration `inputConfig` and that can be used with an audio input stream with this input channel configuration to obtain an audio output stream with the output channel configuration `outputConfig`.

Return Values
Standard

Precondition
— `inputConfig` in `outconf.suppInputConfigs[]`
— Let `i` be the index of `inputConfig` in `outconf.suppInputConfigs[]`
— `outputConfig` in `outconf.inputProcesses[i].pOutputConfigs[]`
— Let `j` be the index of `outputConfig` in outconf.`inputProcesses[i].pOutputConfigs[]`
— `standard != uhOutConf_StdNone`
— `(standard & outconf.inputProcesses[i].pStandards[j]) == standard`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— If (standard == uhOutConf_StdBasic)
  — *pProcesses == outconf.inputProcesses[i].pProcesses[j].basicProcesses
  else if (standard == uhOutConf_StdDolby)
  — *pProcesses == outconf.inputProcesses[i].pProcesses[j].dolbyProcesses
  else if (standard == uhOutConf_StdIncred)
  — *pProcesses == outconf.inputProcesses[i].pProcesses[j].incredProcesses
  else if (standard == uhOutConf_StdSrs)
  — *pProcesses == outconf.inputProcesses[i].pProcesses[j].srsProcesses
  else if (standard == uhOutConf_StdDts)
  — *pProcesses == outconf.inputProcesses[i].pProcesses[j].dtsProcesses

6.1.4.5.1.7 GetEffectRange

Signature
```
uhErrorCode_t GetEffectRange (
    [in] uhOutConf_Standard_t standard,
    [in] uhOutConf_Process_t process,
    [out] pUInt32 pMinEffect,
    [out] pUInt32 pMaxEffect );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and maximum effect amount for the specified process.

Parameters

Table 64

Name	Description
standard	The Standard to which the process process belongs.
process	The process for which the range of effect amount that can be used is requested.
pMinEffect	The minimum effect amount for the basic process process.
pMaxEffect	The maximum effect amount for the basic process process.

Return Values
Standard

Precondition
— standard != uhOutConf_StdNone
— process != uhOutConf_ProcessNone

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
— If (standard == uhOutConf_StdBasic)
   — *pMinEffect == outconf.basicEffectRange[process].min
   — *pMaxEffect == outconf.basicEffectRange[process].max
   else if (standard == uhOutConf_StdDolby)
   — *pMinEffect == outconf.dolbyEffectRange[process].min
   — *pMaxEffect == outconf.dolbyEffectRange[process].max
   else if (standard == uhOutConf_StdIncred)
   — *pMinEffect == outconf.incredEffectRange[process].min
   — *pMaxEffect == outconf.incredEffectRange[process].max
   else if (standard == uhOutConf_StdSrs)
   — *pMinEffect == outconf.srsEffectRange[process].min
   — *pMaxEffect == outconf.srsEffectRange[process].max
   else if (standard == uhOutConf_StdDts)
   — *pMinEffect == outconf.dtsEffectRange[process].min
   — *pMaxEffect == outconf.dtsEffectRange[process].max
```

6.1.4.5.1.8 SetProcess

Signature
```
uhErrorCode_t SetProcess (
    [in] uhOutConf_Config_t inputConfig,
    [in] uhOutConf_Config_t outputConfig,
    [in] uhOutConf_Standard_t standard,
    [in] uhOutConf_Process_t process,
    [in] UInt32 effect );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the process and the associated Standard that should be used whenever an audio signal with the specified channel configuration is present, in order to obtain an output stream with the specified channel configuration.

Parameters

Table 65

Name	Description
`inputConfig`	The input channel configuration for which the process that should be used is set.
`outputConfig`	The output channel configuration that should be obtained when the process `process` is used with audio input signals with the channel configuration `inputConfig`.
`standard`	The Standard to which the process `process` belongs.
`process`	The process that should be used whenever an audio input signal with the specified channel configuration `inputConfig` is present.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
effect	The amount of effect to be used for the process process whenever an audio input signal with the specified channel configuration inputConfig is present.

Return Values
Standard

Precondition
— inputConfig in outconf.suppInputConfigs[]
— Let i be the index of inputConfig in outconf.suppInputConfigs[]
— outputConfig in outconf.inputProcesses[i].pOutputConfigs[]
— Let j be the index of outputConfig in outconf.inputProcesses[i].pOutputConfigs[]
— (standard & outconf.inputProcesses[i].pStandards[j]) == standard
— If (standard == uhOutConf_StdNone)
  — outputConfig == inputConfig
  — process == uhOutConf_ProcessNone
  — effect == 0
  else if (standard == uhOutConf_StdBasic)
  — (process & outconf.inputProcesses[i].pProcesses[j].basicProcesses) == process
  — outconf.basicEffectRange[process].min ≤ effect ≤ outconf.basicEffectRange[process].max
  else if (standard == uhOutConf_StdDolby)
  — (process & outconf.inputProcesses[i].pProcesses[j].dolbyProcesses) == process
  — outconf.dolbyEffectRange[process].min ≤ effect ≤ outconf.dolbyEffectRange[process].max
  else if (standard == uhOutConf_StdIncred)
  — (process & outconf.inputProcesses[i].pProcesses[j].incredProcesses) == process
  — outconf.incredEffectRange[process].min ≤ effect ≤ outconf.incredEffectRange[process].max
  else if (standard == uhOutConf_StdSrs)
  — (process & outconf.inputProcesses[i].pProcesses[j].srsProcesses) == process
  — outconf.srsEffectRange[process].min ≤ effect ≤ outconf.srsEffectRange[process].max
  else if (standard == uhOutConf_StdDts)
  — (process & outconf.inputProcesses[i].pProcesses[j].dtsProcesses) == process
  — outconf.dtsEffectRange[process].min ≤ effect ≤ outconf.dtsEffectRange[process].max

Action
— Modify outconf.prefOutputConfig[i]
— Modify outconf.prefStandard[i]
— Modify outconf.prefProcess[i]
— Modify outconf.prefEffect[i]

Postcondition
— outconf.prefOutputConfig[i] == outputConfig
— outconf.prefStandard[i] == standard
— outconf.prefProcess[i] == process
— outconf.prefEffect[i] == effect

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.5.1.9 GetProcess

Signature

```
uhErrorCode_t GetProcess (
    [in] uhOutConf_Config_t inputConfig,
    [out] puhOutConf_Config_t pOutputConfig,
    [out] puhOutConf_Standard_t pStandard,
    [out] puhOutConf_Process_t pProcess,
    [out] pUInt32 pEffect );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the process and the associated Standard that should be used whenever an audio signal with the specified channel configuration is present. Also returns the output channel configuration that should be obtained when applying this process.

Parameters

Table 66

Name	Description
`inputConfig`	The input channel configuration for which the process that should be used is requested.
`pOutputConfig`	Output parameter returning the output channel configuration that should be obtained when the process *`pProcess` is used with audio input signals with the channel configuration `inputConfig`.
`pStandard`	Output parameter returning the standard to which the process *`pProcess` belongs.
`pProcess`	Output parameter returning the process that should be used whenever an audio input signal with the specified channel configuration `inputConfig` is present.
`pEffect`	Output parameter returning the amount of effect to be used for the process *`pProcess` whenever an audio input signal with the specified channel configuration `inputConfig` is present.

Return Values

Standard

Precondition

— `inputConfig` in `suppInputConfigs[]`

— Let `i` be the index of `inputConfig` in `outconf.suppInputConfigs[]`

Action

None.

Postcondition

— `*pOutputConfig == outconf.prefOutputConfig[i]`

— `*pStandard == outconf.prefStandard[i]`

— `*pProcess == outconf.prefProcess[i]`

— `*pEffect == outconf.prefEffect[i]`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.4.5.1.10 GetActualProcess

Signature

```
uhErrorCode_t GetActualProcess (
    [out] puhOutConf_Config_t pActualInputConfig,
    [out] puhOutConf_Config_t pActualOutputConfig,
    [out] puhOutConf_ValStat_t pStatus,
    [out] puhOutConf_Standard_t pActualStandard,
    [out] puhOutConf_Process_t pActualProcess,
    [out] pUInt32 pActualEffect );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the current process and the associated Standard that is applied to the audio input stream. Also returns the current channel configuration of the audio input stream as well as the current channel configuration of the audio output stream.

Parameters

Table 67

Name	Description
`pActualInputConfig`	Output parameter returning the current channel configuration of the audio input stream.
`pActualOutputConfig`	Output parameter returning the current channel configuration of the audio output stream.
`pStatus`	Output parameter that indicates the validity of the information over the current Standard used `(*pActualStandard)`, over the current process applied `(*pActualProcess)` and over the amount of effect used `(*pActualEffect)`.
`pActualStandard`	Output parameter returning the standard to which the current process *`pActualProcess` belongs.
`pActualProcess`	Output parameter returning the current process that is applied to the audio input stream.
`pActualEffect`	Output parameter returning the current amount of effect used for the current process *`pActualProcess`.

Return Values

Standard

Precondition

`True`

Action

None.

Postcondition

— `*pActualInputConfig == outconf.currentInputConfig`
— `*pActualOutputConfig == outconf.currentOutputConfig`
— `*pStatus == outconf.currentStatus`
— `*pActualStandard == outconf.currentStandard`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— *pActualProcess == outconf.currentProcess
— *pActualEffect == outconf.currentEffect

6.1.4.5.2 uhlOutConfNtf

Qualifiers
— callback

Description
This is the notification interface to be provided by each client that wants to subscribe to notifications of events that are related to changes in the channel configuration of the audio output stream and to changes in the process and the associated Standard applied to the audio input stream.

Interface ID
uuid(27BE787E-6458-4227-8845-7E68F97D1421)

6.1.4.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhOutConf_NtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

6.1.4.5.2.2 OnActualProcessChanged

Signature
```
[async] Void OnActEnabledChanged (
    [in] UInt32 cookie,
    [in] uhOutConf_Config_t inputConfig,
    [in] uhOutConf_Config_t outputConfig,
    [in] uhOutConf_ValStat_t status,
    [in] uhOutConf_Standard_t standard,
    [in] uhOutConf_Process_t process,
    [in] UInt32 effect );
```

Qualifiers
None.

Description
This function is called by the `OutputConfig` whenever it detects a change in the actual process applied to the audio input stream, a change in the actual Standard associated with the process applied and/or a change in the channel configuration of the audio output stream.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 68

Name	Description
cookie	The value that was passed by the client at subscription time.
inputConfig	The channel configuration of the audio input stream.
outputConfig	The channel configuration of the audio output stream.
status	Parameter that indicates the validity of the information over the Standard used (standard), over the process applied (process) and over the amount of effect used (effect).
standard	The Standard to which the current process process belongs.
process	The process applied to the audio input stream.
effect	The amount of effect used for the current process process.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action

Postcondition
True

6.1.5 uhlPinObjects

6.1.5.1 Concepts

6.1.5.1.1 Pin Objects

A Pin Object is an object that is related to an input or output (pin) of a logical component. It is not, as the name might suggest, an object that represents the pin itself. Since inputs and outputs are needed for connecting logical components in a platform, Pin Objects can be considered part of connection management functionality 6.1.1. Pin Objects are relevant when a logical component has multiple inputs or outputs. Clients want to control the functionality belonging to connected inputs and outputs but it will not always be straightforward for the control client to know what object is related to an input or output. Also the number of inputs and outputs may be a diversity parameter and change per connection manager use-case. Using Pin Objects is a method to give the control client access to objects related to an input or output of that logical component.

The Pin Objects interface, described in this document, provides functionality to retrieve the interface pointer belonging to a pin object. Pins of a logical component can be given a pin identifier. By means of this identifier, the interface pointer of the related pin object can be retrieved. Figure 15 shows the concept of pin objects.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

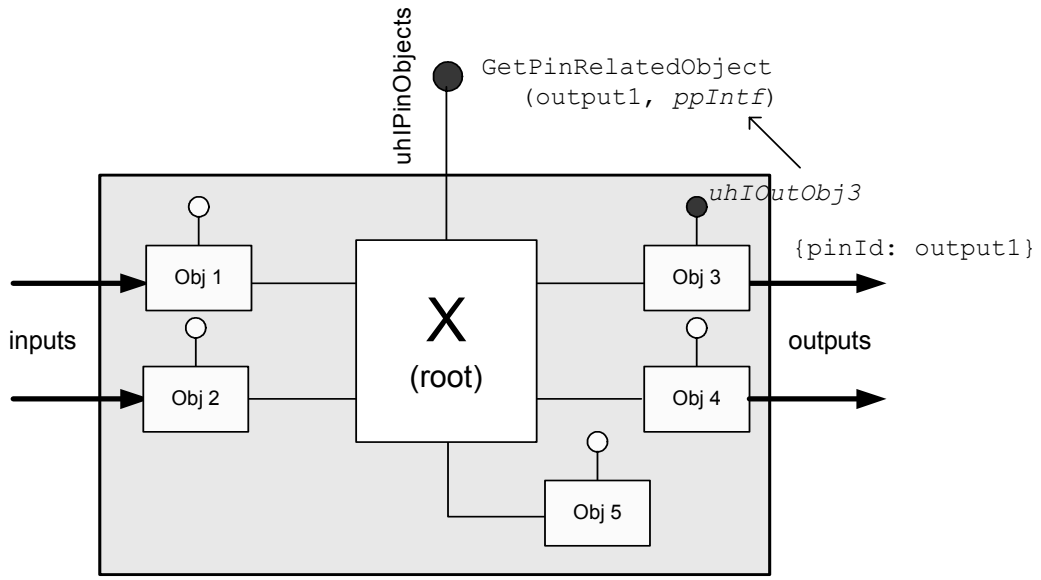


Figure 15 — Pin Objects concept.

The Pin Objects interface can be provided by all logical components that have non-root objects that are related to an input or output (Obj 1 - Obj 4). In general, getting access to a non-root object has to be done via a function on an interface of the root role. This function can be placed on a component-specific interface or the generic Pin Objects interface can be specialized. In Figure 15, Obj 5 is not related to an input or output. To retrieve its interface pointer, a function has to be available on a component specific interface.

The need for pin-related objects becomes clear in case there are multiple identical objects (e.g. Obj 3 and Obj 4) that can only be identified by means of input or output pins. The association of input or output pins with objects depends on how logical components are connected within a platform instance use-case. Pin-related objects are therefore seen as part of connection management functionality and not part of the control functionality of logical components.

6.1.5.1.1.1 Separation of Functionality

The important motivation to use a specialized Pin Objects interface instead of a component specific interface, is to separate the functionality of the logical component (control) and the connection management related functionality to get access to pin-related objects. First, connection management and component control address different functionality and their interface evolution lifetime is very likely not the same. Connection management and control should be independent of each other. Secondly, the actual configuration and the number of inputs or outputs of logical components is determined per use-case and is platform-specific. This dependency is not desirable for logical components since in general it is preferred to make interface suites for logical components platform independent.

Pin Objects also provides the means to separate concerns at the client side. Control clients that need access to the pin-related objects can request the pin objects themselves. However, it is more desirable to use a connection manager client that retrieves the pin-related objects and that passes them to the control clients. This way, control clients do not depend on connection management and the connection manager client does not depend on component specific interfaces. This approach is used for the example of subclause 6.1.5.1.2 (See Figure 16).

For these reasons a generic Pin Objects interface is used and not a specific component interface. Changes in the interface of a logical component or the Pin Objects do not influence each other.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.5.1.1.2 Lifetime

A pin-related object has the same lifetime as the logical component instance of which it is part. During that lifetime a pin object keeps its state. Within a connection manager use-case, the object associated with a given pin identifier remains the same.

6.1.5.1.2 Usage Example

This paragraph shows how a logical component should specialize the Pin Objects Provider. As an example, a simplified interface-role model of a Digital Audio Decoder is used.

Figure 16 shows the interface-role model of the Digital Audio Decoder (DAD). The DAD has an input stream containing a number of audio channels.

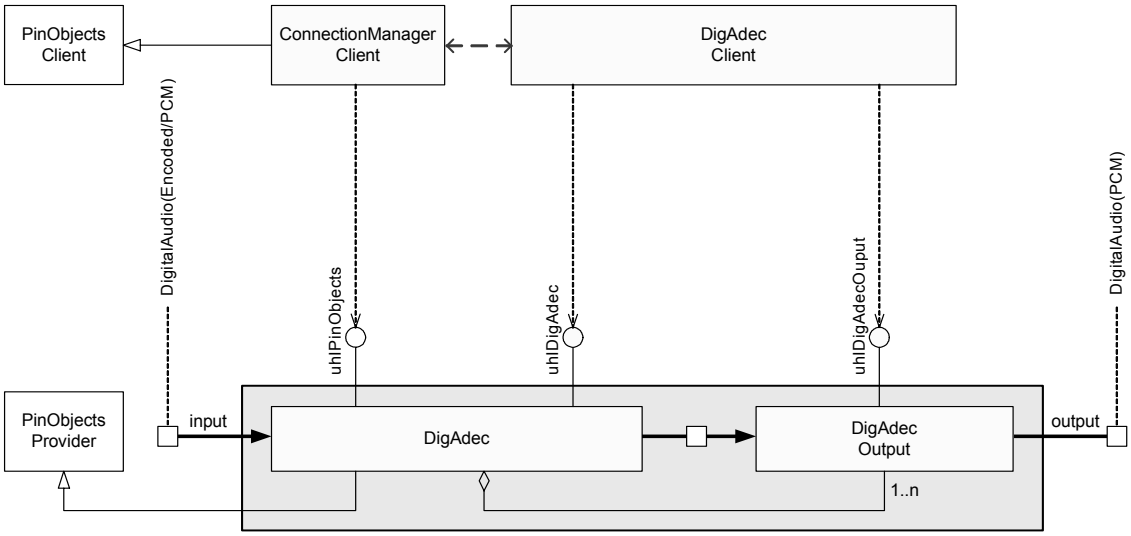


Figure 16 — Example Interface-Role model of Digital Audio Decoder using Pin Objects.

There are a number of outputs for passing a decoded channel (1..n). Every output pin has an associated `DigAdecOutput` object. Since these output objects are all identical, it is not clear which object belongs to a certain output pin. Therefore, a `PinObjectsProvider` role is used. By giving every output pin a unique identifier, the `ConnectionManagerClient` role can request the pin-related object interface pointer for every output pin and pass it to the `DigAdecClient` role.

It is not necessary to give the input pin an identifier since it is clear which object is associated to the input.

6.1.5.1.2.1 Defining Pins

The following indented text gives an example of how to define the pin identifiers. One does not have to define a constant for every input or output. For this example, one can also define a base value and add the output number to it. Within a logical component, pin identifiers must be unique.

Signature

```
const UInt32 uhDigAdec_OutputBase = 0x00000001;
```

Qualifiers

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the constants that identify the output pins of a Digital Audio Decoder.

Constants

Table 69

Name	Description
uhDigAdec_OutputBase	Identified the first output pin of the Digital Audio Decoder.

6.1.5.1.2.2 Initializing Attributes

The document section defining the initial state of a logical component must define all valid pin identifiers and the related objects. For this, specialized attributes of the Pin Objects Provider are used (see subclause 6.1.5.4.1). The following indented paragraph gives an example. nrOutputs is a configurable item of the DAD and its value is defined by the platform instance document.

Objects Created
The following objects are created when the logical component is instantiated:

Table 70

Type	Object	Multiplicity
DigAdec	dec	1
DigAdecOutput	output	nrOutputs

Initial State
The following constraints apply to the initial state of a logical component instance:

- dec.pinIds == {uhDigAdec_OutputBase .. (uhDigAdec_OutputBase + nrOutputs -1)}
- For each i in {0 .. nrOutputs-1}
  - dec.pinObjects[uhDigAdec_OutputBase+i] == output[i]

6.1.5.2 Types & Constants

6.1.5.2.1 Public Types & Constants

None.

6.1.5.2.2 Model Types & Constants

None.

6.1.5.3 Logical Component

6.1.5.3.1 Interface-Role Model

Figure 17 depicts the interface-role model of the Pin Objects LC. It shows the roles and interfaces the Pin Objects LC and its client can have.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

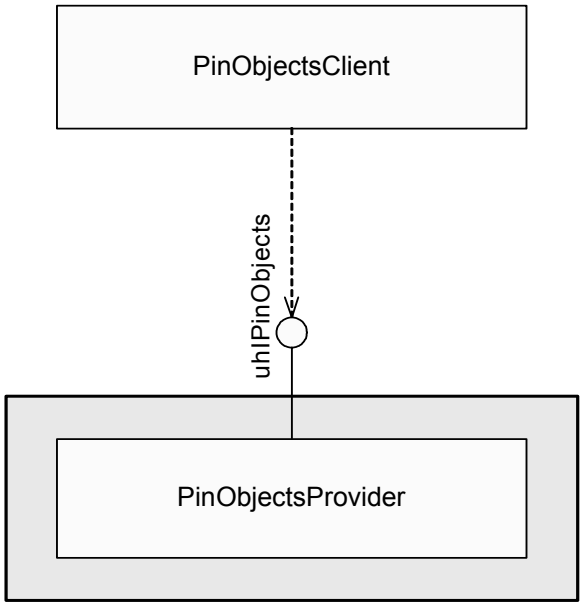


Figure 17 — Interface-Role Model of the Pin Objects provider.

The Pin Objects LC (grey box) consists of the `PinObjectsProvider` role. This role provides the interface `uhIPinObjects` to request interface pointers of objects related to input or output pins. The control interface is meant for usage by the client `PinObjectsClient`. From a design point of view, it is desirable that the `PinObjectsClient` is implemented by a control specialized for connection management, e.g. `ConnectionManagerClient` (see subclause 6.1.5.1.1.1).

6.1.5.3.2 Diversity

6.1.5.3.2.1 Provided Interfaces

Table 71

Role	Interface	Presence
`PinObjectsProvider`	`uhIPinObjects`	mandatory

6.1.5.3.2.2 Configurable Items

Table 72

Role	Attribute
`PinObjectsProvider`	`pinIds`
	`pinObjects`

The configurable items will be given values at instantiation of the logical component that specializes Pin Objects.

6.1.5.3.2.3 Constraints

— None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.5.3.3 Instantiation

6.1.5.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 73

Type	Object	Multiplicity
`PinObjectsProvider`	`pinObj`	1

6.1.5.3.3.2 Initial State

No constraints apply to the initial state of a logical component instance. Constraints are determined by the logical component that specializes the `PinObjectsProvider`.

6.1.5.3.4 Execution Constraints

No execution constraints apply to the Pin Objects logical component. Execution constraints are determined by the logical component that specializes the `PinObjectsProvider`.

6.1.5.4 Roles

6.1.5.4.1 PinObjectsProvider

Signature

```
role PinObjectsProvider {
    const setof{UInt32} pinIds;
    const Object        pinObjects[UInt32];
}
```

Qualifiers

— root

Description

The `PinObjectsProvider` provides functionality to retrieve interface pointers of pin-related objects.

Independent Attributes

Table 74

Name	Description
`pinIds`	The set of pin identifiers that are associated with pin-related objects. Which pin identifiers are valid is defined in the suite of the logical component that specializes Pin Objects and possibly also by the platform instance document. E.g. when a logical component has a configurable number of output pins, a base identifier will be defined in the logical component specification. The platform instance document will define the number of output pins (diversity). Together this results in a valid identifier range: `{baseId .. (baseId + nrOutputs -1) }`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
`pinObjects`	The mapping from pin identifiers in the set `pinIds` to the objects associated with those pin identifiers. The logical component that specialized `PinObjectsProvider` will create these objects.

Dependent Attributes
None.

Invariants
None.

Instantiation
A `PinObjectsProvider` is always instantiated as part of a Pin Objects logical component; see subclause 6.1.5.3.3. A `PinObjectsProvider` is never instantiated as a separate entity but always as part of the root role of the logical component that specializes it.

Streaming Behavior
None.

Active Behavior
None.

6.1.5.4.2 PinObjectsClient

Signature
role `PinObjectsClient {}`

Qualifiers
— actor

Description
A `PinObjectsClient` represents a user of the Pin Objects provider.

6.1.5.5 Interfaces

6.1.5.5.1 uhlPinObjects

Qualifiers
— None.

Description
This interface provides functionality to request the interface pointer of an object associated with a logical component pin.

Interface ID
`uuid( 518B12EC-BAAA-4eb7-9A65-5068B399669B )`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks
In the function specification below we use the name `pinObj` to refer to the instance of role `PinObjectsProvider` providing this interface; see subclause 6.1.5.3.3.1.

6.1.5.5.1.1 GetPinRelatedObject

Signature

```
uhErrorCode_t GetPinRelatedObject (
       [in]  UInt32                    pinId,
       [in]  uhRefIid_t                riid,
       [out,iid_is(riid)] uhIUnknown   **ppIntf);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the interface pointer to the object related to the pin having the given pin identifier.

Parameters

Table 75

Name	Description
`pinId`	Identifier of the pin for which the interface pointer is requested
`riid`	Pointer to the IID of the interface (pointer) to be returned.
`ppIntf`	Output parameter returning the interface pointer to the object associated with the given pin.

Return Values
Standard.

Precondition
— `pinId` is an element of `pinObj.pinIds`.
— `pinObj.pinObjects[pinId]` has an interface `I` of type `*riid`

Action
— None.

Postcondition
— `*ppIntf` == pointer to `I`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.6 RclUnknown / UhlUnknown

6.1.6.1 Concepts

6.1.6.1.1 Introduction

The M3W API has been designed to be compatible with Microsoft COM [Annex B of ISO/IEC 23004-1]. This implies among other things that all run-time objects that provide a M3W API interface, referred to as "M3W objects", also have to provide an interface that is known in COM as "IUnknown". This interface serves three main purposes:

a) It provides a function (`QueryInterface`) to navigate from one interface of an object to another using the "interface ID" (IID) of the interfaces. The IID is a globally unique identifier that can be seen as the "type" of an interface. The IIDs of the interfaces provided by an object are all different.

b) It provides functions (`AddRef` and `Release`) to perform reference counting on M3W objects. The reference count of an object indicates how many references to the interfaces of the object exist. If it reaches zero, the object is no longer reachable and its resources may be freed.

c) It defines the identity of an object. COM does not provide direct access to the identity of an object. The interface-based programming model of COM implies that the identity of objects has to be established by comparing interface pointers. COM allows different pointers to point to the same interface of an object *except* for the `IUnknown` interface pointer which is guaranteed to be unique. Hence two objects are identical if and only if their `IUnknown` interface pointers are identical.

The M3W equivalent of the `IUnknown` interface is called `rcIUnknown` which is also called `uhIUnknown` in the specification of the audio and video interface suites. Like `IUnknown`, this interface is "inherited" by all other UHAPI interfaces. This implies that the functions `QueryInterface`, `AddRef` and `Release` are available in *all* interfaces. In order to call these functions on an object there is no need to first navigate to the `rcIUnknown / uhIUnknown` interface of the object.

Note that users of M3W API interfaces, referred to as "M3W API clients", do not necessarily have to provide the `rcIUnknown / uhIUnknown` interface, unless of course they also provide interfaces that are part of the M3W API such as notification interfaces.

6.1.6.2 Types & Constants

6.1.6.2.1 Public Types & Constants

6.1.6.2.1.1 Error Codes

Signature

```
const uhErrorCode_t UH_ERR_UNKNOWN_NO_INTERFACE = 0x00000001;
```

Qualifiers

— error-codes

Description

This section defines the error codes specific to `uhIUnknown`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Constants

Table 76

Name	Description
UH_ERR_UNKNOWN_NO_INTERFACE	Indicates that a requested interface is not available.

6.1.6.2.1.2 uhIid_t

Signature

```
typedef struct _uhIid_t {
    UInt32 data1;
    UInt16 data2;
    UInt16 data3;
    UInt8  data4[8];
} uhIid_t, *puhIid_t;
```

Qualifiers
None.

Description
The values of uhIid_t represent interface IDs (IIDs), i.e. globally unique identifiers (GUIDs) that uniquely identify “interface types”.

Constraints
— None.

Remarks
— The above definition conforms to the COM definition of IIDs.

— In specifications values of this type are usually represented as 32-digit (128-bit) hexadecimal numbers in a format exemplified below:
uuid(5697C761-5E73-11d6-9A8B-00065B6400D5)

6.1.6.2.1.3 uhReflid_t

Signature

```
typedef const uhIid_t* const uhRefIid_t;
typedef uhRefIid_t *puhRefIId_t;
```

Qualifiers
None.

Description
The values of uhRefIid_t represent pointers to IIDs.

Constraints
— Each value of type uhRefIid_t points to a valid IID.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks

— The above constraint implies that in-parameters of type `uhRefIid_t` in functions always point to valid IIDs. This constraint is therefore not repeated in the preconditions of these functions.

— The reason to introduce this type is that the size of IIDs is 128 bits, which makes it more efficient to pass pointers to IIDs rather than the IIDs themselves.

6.1.6.2.2 Model Types & Constants

6.1.6.2.2.1 Interface

Signature

```
entity Interface {
    const uhIid_t iid;
    const Object  provider;
    UInt32        refCount;
}
```

Qualifiers

None.

Description

The entities of this type are used to model the interfaces of objects, where “interface” is meant in the sense of “interface instance”. Each interface (instance) has a unique IID which can be interpreted as the “interface type”, is provided by a unique object and has a reference count associated with it.

Independent Attributes

Table 77

Name	Description
`iid`	The IID of the interface.
`provider`	The object providing the interface.
`refCount`	The reference count associated with the interface.

Constraints

— None.

Remarks

— According to the COM specification [27] reference counting is done per interface rather than per object. This implies that the number of times Release is called on an interface may not exceed the number of times `AddRef` is called on the same interface. The reference count of an object is the sum of the reference counts of the interfaces provided by the object; see also *Appendix A*.

— The “interfaces” defined by the type Interface should not be confused with “interface pointers”, which are implementation entities pointing to interfaces.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.6.3 Logical Component

6.1.6.3.1 Interface-Role Model

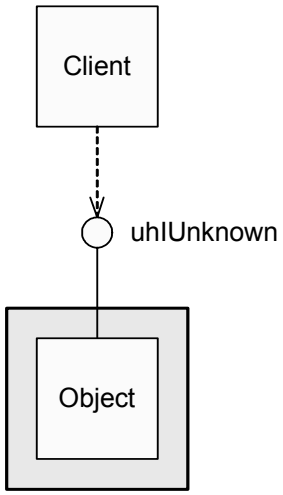


Figure 18 — Interface Role Model

The roles Object and Client in the interface-role model represent “M3W objects” (objects that provide M3W API interfaces) and “M3W API clients” (objects that use M3W API interfaces), respectively. The interface-role model expresses that each M3W object should provide the `rcIUnknown / uhIUnknown` interface and each M3W API client is allowed to use it.

6.1.6.3.2 Diversity

6.1.6.3.2.1 Provided Interfaces

Table 78

Role	Interface	Presence
`Object`	`rcIUnknown / uhIUnknown`	mandatory

6.1.6.3.2.2 Configurable Items

Table 79

Role	Attribute
`Object`	`iidSet`

The `rcIUnknown / uhIUnknown` interface is a generic interface. The set of IIDs for which the `QueryInterface` function will return “success” can be seen as a configuration parameter of this interface that is set at creation time of an object. This configuration parameter is modelled by the attribute `iidSet`.

6.1.6.3.2.3 Constraints

— For each Object x
  — The IID of `rcIUnknown / uhIUnknown` (see 6.1.6.5.1) is an element of `iidSet`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.6.3.3 Instantiation

6.1.6.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 80

Type	Object	Multiplicity
`Object`	`obj`	1
`setof{Interface}`	`intfs`	any

Here intfs represents the initial collection of interfaces of obj. How many interfaces are created is implementation freedom hence the multiplicity (number of elements) of the set intfs is specified as “any”.

6.1.6.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

- For each *i* in `intfs`
  - `i.iid` in `obj.iidSet`
  - `i.provider == obj`
- There is at least one interface i in intfs such that `i.refCount > 0`.

Remarks

- Note that the constraints imply that `obj.interfaces == intfs`.

- As specified above, an object is instantiated together with a non-empty set of interfaces. At least one of these interfaces has a reference count > 0, hence the reference count of the object is > 0 as well. A typical implementation will create interface instances for *all* IIDs in iids, but this is not required. New interface instances can e.g. be created by `QueryInterface`.

6.1.6.3.4 Execution Constraints

No execution constraints apply to the `rcIUnknown / uhIUnknown` logical component; the `rcIUnknown / uhIUnknown` interface is thread-safe.

6.1.6.4 Roles

6.1.6.4.1 Object

Signature

```
role Object {
    const setof{uhIid_t} iidSet;
    setof{Interface}     interfaces;
}
```

Qualifiers

None.

Description

This role models general M3W objects, i.e. objects that provide the `rcIUnknown / uhIUnknown` interface.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 81

Name	Description
`iidSet`	The set of IIDs of the interfaces provided by the object, determining for which IIDs the `QueryInterface` function will be successful. The value of `iidSet` is constant because it is set at object instantiation time and does not change dynamically (in accordance with Microsoft COM).

Dependent Attributes

Table 82

Name	Description
`interfaces`	The set of interfaces of the object, i.e. the set of all instances i of type Interface such that `i.provider` **== this**. The value of this attribute is not constant because interface instances may be created dynamically e.g. due to the creation of interface "proxies" when remotely accessing interfaces. Hence there may be multiple interface instances in interfaces with the same IID.

Invariants

— `iidSet` contains the IID of `rcIUnknown / uhIUnknown` (see 6.1.6.5.1).

Instantiation

See 6.1.6.3.3.

Streaming Behavior

None.

Active Behavior

None.

Remarks

— The reference count of an object can be defined as the sum of the reference counts of its interfaces. In a typical implementation the resources associated with an object, such as the occupied storage, will be freed when the reference count of the object reaches zero. This is not modeled in the specification because it is an implementation issue. From the specification perspective an object lives forever. If the reference count of an object is zero, the reference counts of all of its interfaces are zero and the effect of any call of a function in one of these interfaces will be undefined. In other words, the object no longer serves any useful purpose.

— Each role that is specified as a provider of a M3W API interface is assumed to specialize the Object role. This relation is not normally indicated in interface-role diagrams; see Annex D.

6.1.6.4.2 Client

Signature

```
role Client {}
```

Qualifiers

— actor

Description

This role models users of the `rcIUnknown / uhIUnknown` interface.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks
Each role that is specified as a user of a M3W API interface is assumed to specialize the Client role. This relation is not normally indicated in interface-role diagrams; see Annex D.

6.1.6.5 Interfaces

6.1.6.5.1 rclUnknown / uhlUnknown

Qualifiers
None.

Description
The `rcIUnknown / uhIUnknown` interface is a generic interface that is provided by all objects and inherited by all other interfaces. It has three main functions:

- providing interface navigation functionality;
- providing reference counting functionality;
- defining the identity of an object.

Interface ID
`uuid( 00000000-0000-0000-C000-000000000046 )`

Remarks
- The fact that `rcIUnknown / uhIUnknown` is "inherited" by all other interfaces of the M3W API implies that the functions `QueryInterface`, `AddRef` and `Release` are available in *all* interfaces. In order to call these functions on an object there is no need to first navigate to the `rcIUnknown / uhIUnknown` interface of the object.
- As specified below, all functions in `rcIUnknown / uhIUnknown` have precondition "`refCount > 0`". This is to indicate that no guarantees are given when such a function is called on an interface whose reference count is zero. The precondition "`refCount > 0`" is implicit in the precondition of each UHAPI function; see Annex D. The specification of the `rcIUnknown / uhIUnknown` interface is the only place where this condition is made explicit.

6.1.6.5.1.1 QueryInterface

Signature
```
uhErrorCode_t QueryInterface(
   [in]  uhRefIid_t riid,
   [out] pVoid      *pUnk
);
```

Qualifiers
- Synchronous

Description
Tests whether the current object provides an interface with a specified IID. If the object provides the interface, a pointer to an interface with the specified IID is returned. If the object does not provide the interface, a null pointer is returned.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 83

Name	Description
riid	Pointer to the IID of the interface (pointer) to be returned.
pUnk	Output parameter returning either a pointer to an interface of type *iid or Null.

Return Values

Table 84

Name	Description
UH_OK	Indicates that the object provides an interface with the specified IID.
UH_ERR_UNKNOWN_NO_INTERFACE	Indicates that the object does not provide an interface with the specified IID.

Precondition

— refCount > 0
— Let iid = *riid

Case 1 *(Object provides an interface of type* iid**)**

— iid **in** iidSet

Action

— Let intf be an existing or new Interface such that
  — intf.iid == iid
  — intf.provider == **this**
— Modify intf.refCount

Postcondition

— *pUnk == &intf
— intf.refCount == intf.refCount' + 1
— **result** == UH_OK

Case 2 *(Object does not provide an interface of type* iid*)*

— !(iid **in** iidSet)

Action

None.

Postcondition

— *pUnk == Null
— **result ==** UH_ERR_UNKNOWN_NO_INTERFACE

Remarks

— Note that "refCount" in the precondition refers to the refCount attribute of the interface on which QueryInterface is called; it is not an attribute of the object.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— As implied by Case 1 of the specification, `QueryInterface` does not necessarily return the same interface pointer when asked for the same interface multiple times. The only exception to this is a query for `rcIUnknown / uhIUnknown`. Within the context of a single process the pointer to the `rcIUnknown / uhIUnknown` interface of an object is unique, i.e. exactly one interface instance of type `rcIUnknown / uhIUnknown` is accessible in the process. This constraint is implied by the COM compatibility and allows pointers to `rcIUnknown / uhIUnknown` to be used for comparing the identities of objects. The constraint is not explicitly modeled in the specification because it requires modeling part of the execution architecture as well.

— Note that `QueryInterface` increments the reference count of the interface whose pointer it returns. If an application no longer needs to use the interface retrieved by a call to `QueryInterface`, it must therefore call the Release method for that interface to free it.

6.1.6.5.1.2 AddRef

Signature
```
UInt32 AddRef();
```

Qualifiers
— synchronous

Description
Increments the reference count of the interface on which `AddRef` is called.

Parameters
`None.`

Return Values
See the postcondition.

Precondition
```
refCount > 0
```

Action
— Modify `refCount`

Postcondition
— `refCount == refCount' + 1`
— result > 0

Remarks
— Note that "`refCount`" in the precondition refers to the `refCount` attribute of the interface on which `AddRef` is called; it is not an attribute of the object.

— Even though the COM specification describes the value returned by `AddRef` as "the resulting value of the reference count", it also specifies that absolutely no meaning can be assigned to this return value.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.6.5.1.3 Release

Signature
`UInt32 Release();`

Qualifiers
— synchronous

Description
Decrements the reference count of the interface on which Release is called.

Parameters
`None.`

Return Values
See the postcondition.

Precondition
— `refCount` > 0

Action
— Modify `refCount`

Postcondition
— `refCount == refCount' - 1`
— If (**result** == 0) then `refCount` == 0

Remarks
— Note that "`refCount`" in the precondition refers to the `refCount` attribute of the interface on which Release is called; it is not an attribute of the object.

— When `refCount` becomes zero as a consequence of calling Release no more references to the interface are supposed to exist, i.e. all references to the interface are "invalid". Implementations can therefore choose to free the resources associated with the interface. Note that the resources associated with an object can only be freed if the reference counts of all of its interfaces are zero.

— Even though the COM specification describes the value returned by Release as "the resulting value of the reference count", it also specifies that this return value only has meaning if it is zero. In that case, all references to the interface can be considered invalid, as expressed by the last part of the postcondition.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7 Audio interface suites

7.1 Broadcast Audio decoding

7.1.1 UhlAnaAdec

7.1.1.1 Concepts

7.1.1.1.1 Introduction

This subclause is structured in the following way. First an overview over common analog audio broadcasting standards is given. Then the basic working principle of the Analog Audio Decoder is explained. In this 2nd part especially the relation between input signals, decoding and output signals is described.

7.1.1.1.2 Analog Audio Broadcasting Standards

In analog broadcasting, the audio signal broadcasted can be classified by two criteria's.

The first criteria is the Sound System, which is used to encode the information. The second criteria are the Sound Properties, which are more defining the additional capability of the signal. The combination of the two is unambiguously identifying a broadcasted audio signal.

Sound Standard can be one of the following:

Table 85 — Sound standards

Sound Standard	
AM mono	Analog signal.
FM mono	Analog signal.
2CS	Analog signal.
FM Korea	Analog signal.
BTSC	Analog signal.
EIAJ	Analog signal.
NICAM with AM mono	Digital extension to AM mono audio standard.
NICAM with FM mono	Digital extension to FM mono audio standard.

Note that the Sound Standards should not be confused with TV systems. A TV system deals with the spectral location of the sound carriers, the sound standard with the modulation and coding techniques. Sound standards and TV systems are more or less independent, though in practice, certain combinations are fixed (such as e.g. the AM sound standard with TV system L).

Sound Properties on the other hand can be one of the following:

Table 86 — Sound properties

Sound Properties	
None	No additional property, the audio signal is fully defined by the Sound Standard.
Mono	The audio stream contains a mono signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Sound Properties

Stereo	The audio stream contains a stereo signal.
Dual	The audio stream contains a 2nd language.
Unrelated	The audio stream contains information, which is unrelated to the video information.
Data 352 kbit/s	The audio stream contains data information at a rate of 352 kbit/s.
Data 704 kbit/s	The audio stream contains data information at a rate of 704 kbit/s.

Nevertheless, not all combinations of Sound Standards and Sound Properties are possible. Table 87 shows all defined combinations. Since Table 87 is a full list with all the possible audio signals an Analog Audio Decoder has to deal with, we linked every combination with an Id-Number that is used as a reference further on in this document.

Table 87 — Audio Signal Classification

Audio Signals

Id	Name
1	Invalid
2	Unknown
3	AM mono
4	FM mono
5	2CS Stereo
6	2CS Dual
7	FM Korea Stereo
8	FM Korea Dual
9	BTSC Stereo
10	BTSC Mono + SAP
11	BTSC Stereo + SAP
12	EIAJ Stereo
13	EIAJ Dual
14	AM mono + NICAM Mono
15	AM mono + NICAM Stereo
16	AM mono + NICAM Dual
17	AM mono + NICAM Data only (704 kbit/s)
18	AM mono + NICAM Mono + Data (352 kbit/s)
19	AM mono + unrelated NICAM Mono
20	AM mono + unrelated NICAM Stereo
21	AM mono + unrelated NICAM Dual
22	AM mono + unrelated NICAM Data only (704 kbit/s)
23	AM mono + unrelated NICAM Mono + Data (352 kbit/s)
24	FM mono + NICAM Mono
25	FM mono + NICAM Stereo
26	FM mono + NICAM Dual
27	FM mono + NICAM Data only (704 kbit/s)
28	FM mono + NICAM Mono + Data (352 kbit/s)

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Audio Signals

29	FM mono + unrelated NICAM Mono
30	FM mono + unrelated NICAM Stereo
31	FM mono + unrelated NICAM Dual
32	FM mono + unrelated NICAM Data only (704 kbit/s)
33	FM mono + unrelated NICAM Mono + Data (352 kbit/s)

The NICAM, BTSC and 2CS sound standards are defined in [8], [9] and [10], respectively.

7.1.1.1.3 Decoding Working Principles

The input of the Analog Audio Decoder is an analog audio stream, encoded with one of the sound standards mentioned above.

On one hand, the Analog Audio Decoder is detecting the sound standard and sound properties of the input stream. This detection may take some time. The decoder can be instructed to detect specific sound standards only, by limiting the set of allowed sound standards.

On the other hand, the Analog Audio Decoder is decoding the input stream to baseband audio.

Depending on the input signal, several independent audio streams may be encoded in the input stream. For example, dual language signals contain two independent audio streams, one for the first language and one for the second language. The Analog Audio Decoder is always decoding all streams contained in the input signal and outputs them all. Selection of one of the stream is not in the scope of the Analog Audio Decoder, but is done downstream by another logical component.

So, the output of the Analog Audio Decoder can be seen as multiple baseband signals multiplexed to one output stream.

7.1.1.2 Types & Constants

7.1.1.2.1 Public Types & Constants

7.1.1.2.1.1 Error Codes

None.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.1.2.1.2 uhAnaAdec_SoundStandard_t

Signature

```
typedef enum _uhAnaAdec_SoundStandard_t {
    uhAnaAdec_SoundStandardInvalid = 0x80000000,
    uhAnaAdec_SoundStandardUnknown = 0x40000000,
    uhAnaAdec_SoundStandardAmMono  = 0x00000002,
    uhAnaAdec_SoundStandardFmMono  = 0x00000004,
    uhAnaAdec_SoundStandard2Cs     = 0x00000008,
    uhAnaAdec_SoundStandardFmKorea = 0x00000010,
    uhAnaAdec_SoundStandardBtsc    = 0x00000020,
    uhAnaAdec_SoundStandardEiaj    = 0x00000040,
    uhAnaAdec_SoundStandardAmNicam = 0x00000080,
    uhAnaAdec_SoundStandardFmNicam = 0x00000100
} uhAnaAdec_SoundStandard_t, *puhAnaAdec_SoundStandard_t;
```

Qualifiers

— enum-element

Description

Values of this type represent sound standards.

Values

Table 88

Name	Description
uhAnaAdec_SoundStandardInvalid	The sound standard is temporarily not available.
uhAnaAdec_SoundStandardUnknown	The sound standard is available, yet it could not be determined.
uhAnaAdec_SoundStandardAmMono	AM Mono. The base property of the audio signal is that it contains a Mono signal.
uhAnaAdec_SoundStandardFmMono	FM Mono. The base property of the audio signal is that it contains a Mono signal.
uhAnaAdec_SoundStandard2Cs	Two-Carrier System (also known as German Stereo).
uhAnaAdec_SoundStandardFmKorea	South Korea - which uses the NTSC standard - has adopted a modified version of the Two-Carrier System (2CS).
uhAnaAdec_SoundStandardBtsc	BTSC.
uhAnaAdec_SoundStandardEiaj	EIAJ.
uhAnaAdec_SoundStandardAmNicam	NICAM with AM. The base property of the audio signal is that the analog audio content is a Mono signal.
uhAnaAdec_SoundStandardFmNicam	NICAM with FM. A base property of the audio signal is that the analog audio content is a Mono signal.

7.1.1.2.1.3 uhAnaAdec_SoundStandardSet_t

Signature

```
typedef Int32 uhAnaAdec_SoundStandardSet_t, *puhAnaAdec_SoundStandardSet_t;
```

Qualifiers

— enum-set

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of sound standards

7.1.1.2.1.4 uhAnaAdec_SoundProperty_t

Signature

```
typedef enum _uhAnaAdec_SoundProperty_t {
    uhAnaAdec_SoundPropertyInvalid      = 0x80000000,
    uhAnaAdec_SoundPropertyUnknown      = 0x40000000,
    uhAnaAdec_SoundPropertyMono         = 0x00000001,
    uhAnaAdec_SoundPropertyStereo       = 0x00000002,
    uhAnaAdec_SoundPropertyDual         = 0x00000004,
    uhAnaAdec_SoundPropertyUnrelated    = 0x00000008,
    uhAnaAdec_SoundPropertyData352Kbits = 0x00000010,
    uhAnaAdec_SoundPropertyData704Kbits = 0x00000020
} uhAnaAdec_SoundProperty_t, *puhAnaAdec_SoundProperty_t;
```

Qualifiers
— enum-element

Description
Values of this type represent additional properties of the input stream of an analog audio decoder besides the sound standard.

Values

Table 89

Name	Description
uhAnaAdec_SoundPropertyInvalid	The sound property is temporarily not available.
uhAnaAdec_SoundPropertyUnknown	The sound property is available, yet it could not be determined.
uhAnaAdec_SoundPropertyMono	An additional property of the input signal is that it contains a mono signal. In case the sound standard of the audio signal is NICAM, the additional property relates to the digital audio content.
uhAnaAdec_SoundPropertyStereo	An additional property of the input signal is that it contains a stereo signal. In case the sound standard of the audio signal is NICAM, the additional property relates to the digital audio content
uhAnaAdec_SoundPropertyDual	An additional property of the input signal is that it contains a mono 2nd language signal. In case the sound standard of the audio signal is NICAM, the additional property relates to the digital audio content
uhAnaAdec_SoundPropertyUnrelated	NICAM with AM mono or FM mono has been detected and the digital audio is unrelated to the video content.
uhAnaAdec_SoundPropertyData352Kbits	NICAM with 352 kbit/s data has been detected; only mono digital audio is available.
uhAnaAdec_SoundPropertyData704Kbits	NICAM with data only has been detected; no digital audio is available.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.1.2.1.5 uhAnaAdec_SoundPropertySet_t

Signature
```
typedef Int32 uhAnaAdec_SoundPropertySet_t, *puhAnaAdec_SoundPropertySet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of additional properties of the input stream of an analog audio decoder besides the sound standard.

7.1.1.2.1.6 uhAnaAdec_Notification_t

Signature
```
typedef enum _uhAnaAdec_Notification_t {
    uhAnaAdec_SubscriptionChanged = 0x00000001,
    uhAnaAdec_AudioSignalChanged  = 0x00000002,
    uhAnaAdec_StableOutput        = 0x00000004
} uhAnaAdec_Notification_t, *puhAnaAdec_Notification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAnaAdecNtf`.

Values

Table 90

Name	Description
`uhAnaAdec_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhAnaAdec_AudioSignalChanged`	Represents `OnAudioSignalChanged`.
`uhAnaAdec_StableOutput`	Represents `OnStableOutput`.

7.1.1.2.1.7 uhAnaAdec_Notifications_t

Signature
```
typedef Int32 uhAnaAdec_Notifications_t, *puhAnaAdec_Notifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhAnaAdec`_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Constraints
None.

7.1.1.2.1.8 uhAnaAdec_AllNotifications

Signature
```
const uhAnaAdec_Notifications_t uhAnaAdec_AllNotifications = 0x00000007;
```

Qualifiers
None.

Description
Defines the set of all values of type uhAnaAdec_Notification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

7.1.1.2.2 Model Types & Constants

None.

7.1.1.3 Logical Component

7.1.1.3.1 Interface-Role Model

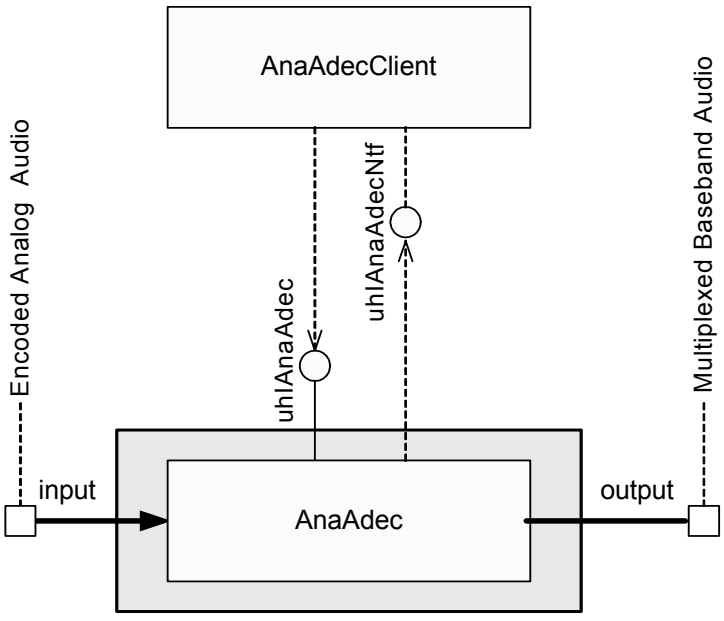


Figure 19 — Interface Role Model

Figure 19 depicts the interface-role model of the Analog Audio Decoding Logical Component. It shows the interfaces and the roles involved with this component.

An Analog Audio Decoding Logical Component contains a Decoder (represented by the AnaAdec role).

Each decoder has its own control interface (uhIAnaAdec). This control interface is meant for use by the regular client (represented by the AnaAdecClient role).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Clients can optionally provide a call-back interface (`uhIAnaAdecNtf`) allowing them to subscribe to notifications of events in the decoder.

7.1.1.3.2 Diversity

7.1.1.3.2.1 Provided Interfaces

Table 91

Role	Interface	Presence
`AnaAdec`	`uhIAnaAdec`	mandatory
`AnaAdecClient`	`uhIAnaAdecNtf`	optional

7.1.1.3.2.2 Configurable Items

Table 92

Role	Attribute
`AnaAdec`	`supportedStandards`

7.1.1.3.2.3 Constraints

— `!(supportedStandards & uhAnaAdec_SoundStandardInvalid)`
— `!(supportedStandards & uhAnaAdec_SoundStandardUnknown)`

7.1.1.3.3 Instantiation

7.1.1.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 93

Type	Object	Multiplicity
`AnaAdec`	`dec`	1

7.1.1.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— `dec.allowedStandards == dec.supportedStandards`
— `dec.detectedStandard == uhAnaAdec_SoundStandardInvalid`
— `dec.detectedProperties == uhAnaAdec_SoundPropertyInvalid`
— `dec.stableOutput == False`

7.1.1.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.1.4 Roles

7.1.1.4.1 AnaAdec

Signature

```
role AnaAdec {
    const uhAnaAdec_SoundStandardSet_t   supportedStandards;
    uhAnaAdec_SoundStandardSet_t         allowedStandards;
    uhAnaAdec_SoundStandard_t            detectedStandard;
    uhAnaAdec_SoundPropertySet_t         detectedProperties;
    Bool                                 stableOutput;
}
```

Qualifiers

— root

Description

An `AnaAdec` decodes an encoded analog audio input stream and provides the decoded programs. It autonomously detects the sound standard of the input stream from a set of allowed sound standards and provides notifications of various changes that occur in the input stream.

Independent Attributes

Table 94

Name	Description
`supportedStandards`	The set of sound standards that the decoder is able to detect.
`allowedStandards`	The set of sound standards that the decoder is allowed to detect.
`detectedStandard`	The sound standard of the input stream as detected by the decoder.
`detectedProperties`	The additional sound properties of the input stream besides the sound standard as detected by the decoder.
`stableOutput`	Indicates whether the analog audio decoder is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream (`True` if this is the case, `False` otherwise).

Invariants

— `(allowedStandards & supportedStandards) == allowedStandards`
— `(detectedStandard & (allowedStandards | uhAnaAdec_SoundStandardInvalid | uhAnaAdec_SoundStandardUnknown)) == detectedStandard`
— if `(detectedStandard == uhAnaAdec_SoundStandardInvalid)`
  — `detectedProperties == uhAnaAdec_SoundPropertyInvalid`
— if `(detectedStandard == uhAnaAdec_SoundStandardUnknown)`
  — `detectedProperties` **==** `uhAnaAdec_SoundPropertyUnknown`

Instantiation

An `AnaAdec` is always instantiated as part of an Analog Audio Decoder; see subclause 7.1.1.3.3.

Streaming Behavior

The decoder continually observes the input stream of the Analog Audio Decoder.

When an input stream is detected, it will try to detect the sound standard of the input stream and set the value of `detectedStandard` accordingly.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The decoder will only detect standards that are in the set `allowedStandards`.

As soon as a sound standard *s* has been detected, the value of `detectedStandard` is set to *s* and the input stream is decoded.

When the sound standard of the input signal is not in the set `allowedStandards, or could not be determined`, the value of `detectedStandard` is set to uh`AnaAdec_SoundStandardUnknown`; no output data is produced.

`Active Behavior`
The `AnaAdec` has standard notification behavior (see `uhIAnaAdecNtf`), apart from the following events.

Table 95

Event	Action
The analog audio decoder is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream. (`dec.stableOutput == False` when this event occurs).	`dec.stableOutput = True` — notify `OnStableOutput(*, True)`
The analog audio decoder is no longer outputting a stable (decoded) audio stream (`dec.stableOutput == True` when this event occurs).	`dec.stableOutput = False` — notify `OnStableOutput(*, False)`

`Execution Constraints`
None.

`Remarks`
The `allowedStandards` attribute is useful for countries/areas where certain sound standards are not used. It can speed up the detection of the right sound standard and it can prevent a sound standard from being detected incorrectly.

7.1.1.4.2 AnaAdecClient

`Signature`
`role` `AnaAdecClient {}`

`Qualifiers`
— actor

`Description`
An Analog Audio Decoder Client represents a user of the `uhIAnaAdec` interface.

`Execution Constraints`
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.1.5 Interfaces

7.1.1.5.1 uhIAnaAdec

Qualifiers
None.

Description
This interface of a decoder provides functionality to control the decoding of audio programs, to retrieve properties of the decoder input stream and to subscribe/unsubscribe to events notified by the decoder. The client can control the decoding by restricting the sound standards to be used by the decoder.

Interface ID
uuid(1BCA5EC3-27FB-4972-91A3-A1DC0C978DA1)

Execution Constraints
None.

7.1.1.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAnaAdecNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAnaAdec_Notifications_t notifs );
```

Qualifiers
— subscribe-function

7.1.1.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAnaAdecNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAnaAdec_Notifications_t notifs );
```

Qualifiers
— unsubscribe-function

7.1.1.5.1.3 GetSuppStandards

Signature
```
uhErrorCode_t GetSuppStandards (
    [out] puhAnaAdec_SoundStandardSet_t pSupportedStandards );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the set of sound standards supported by the decoder.

Parameters

Table 96

Name	Description
pSupportedStandards	Output parameter returning the set of sound standards supported by the decoder.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pSupportedStandards == dec.supportedStandards

7.1.1.5.1.4 SetAllowedStandards

Signature
```
uhErrorCode_t SetAllowedStandards (
    [in] uhAnaAdec_SoundStandardSet_t allowedStandards );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the set of sound standards that the decoder is allowed to detect.

Parameters

Table 97

Name	Description
allowedStandards	The set of sound standards that the decoder is allowed to detect.

Return Values
Standard.

Precondition
— (allowedStandards & dec.supportedStandards) == allowedStandards

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify dec.allowedStandards
— If !(dec.detectedStandard & allowedStandards)
  — Modify dec.detectedStandard
  — Modify dec.detectedProperties

Postcondition
— dec.allowedStandards == allowedStandards
— if (dec.detectedStandard' == uhAnaAdec_SoundStandardInvalid)
  — dec.detectedStandard == uhAnaAdec_SoundStandardInvalid
  — dec.detectedProperties == uhAnaAdec_SoundPropertyInvalid
— else if !(dec.detectedStandard' & allowedStandards)
  — dec.detectedStandard == uhAnaAdec_SoundStandardUnknown
  — dec.detectedProperties == uhAnaAdec_SoundPropertyUnknown

Remarks
SetAllowedStandards is taking effect immediately. I.e. if the previously detected standard is not allowed any more, then dec.detectedStandard will be reset to uhAnaAdec_SoundStandardUnknown and detection will restart to find an alternative standard.

7.1.1.5.1.5 GetAllowedStandards

Signature
```
uhErrorCode_t GetAllowedStandards (
    [out] puhAnaAdec_SoundStandardSet_t pAllowedStandards );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the set of sound standards that the decoder is allowed to detect.

Parameters

Table 98

Name	Description
pAllowedStandards	Output parameter returning the set of sound standards that the decoder is allowed to detect.

Return Values
Standard.

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
```
*pAllowedStandards == dec.allowedStandards
```

7.1.1.5.1.6 GetAudioSignal

Signature
```
uhErrorCode_t GetAudioSignal (
    [out] puhAnaAdec_SoundStandard_t pSoundStandard,
    [out] puhAnaAdec_SoundPropertySet_t pSoundProperties );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns the sound standard and additional sound properties of the input signal as detected by the decoder.

Parameters

Table 99

Name	Description
`pSoundStandard`	Output parameter returning the sound standard of the input signal as detected by the decoder.
`pSoundProperties`	Output parameter returning additional sound properties of the input signal of the decoder besides the sound standard.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
- `*pSoundStandard == dec.detectedStandard`
- `*pSoundProperties == dec.detectedProperties`

Remarks
- Sound standard and audio program information are combined in this function, thus providing synchronous, consistent information on the input stream.
- The following table defines the Sound Standard and additional Sound Property sets for all the audio signal (as defined in subclause 7.1.1.1.1):

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 100

Audio Signal		Sound Standard	Additional Sound Property set
Id	**Name**		
1	Invalid	uhAnaAdec_SoundStandardInvalid	uhAnaAdec_SoundPropertyInvalid
2	Unknown	uhAnaAdec_SoundStandardUnknown	uhAnaAdec_SoundPropertyUnknown
3	AM mono	uhAnaAdec_SoundStandardAmMono	uhAnaAdec_SoundPropertyUnknown
4	FM mono	uhAnaAdec_SoundStandardFmMono	uhAnaAdec_SoundPropertyUnknown
5	2CS Stereo	uhAnaAdec_SoundStandard2Cs	uhAnaAdec_SoundPropertyStereo
6	2CS Dual	uhAnaAdec_SoundStandard2Cs	uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyDual
7	FM Korea Stereo	uhAnaAdec_SoundStandardFmKorea	uhAnaAdec_SoundPropertyStereo
8	FM Korea Dual	uhAnaAdec_SoundStandardFmKorea	uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyDual
9	BTSC Stereo	uhAnaAdec_SoundStandardBtsc	uhAnaAdec_SoundPropertyStereo
10	BTSC Mono + SAP	uhAnaAdec_SoundStandardBtsc	uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyDual
11	BTSC Stereo + SAP	uhAnaAdec_SoundStandardBtsc	uhAnaAdec_SoundPropertyStereo | uhAnaAdec_SoundPropertyDual
12	EIAJ Stereo	uhAnaAdec_SoundStandardEiaj	uhAnaAdec_SoundPropertyStereo
13	EIAJ Dual	uhAnaAdec_SoundStandardEiaj	uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyDual
14	AM mono + NICAM Mono	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyMono
15	AM mono + NICAM Stereo	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyStereo
16	AM mono + NICAM Dual	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyDual
17	AM mono + NICAM Data only (704kbit/s)	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyData704Kbits
18	AM mono + NICAM Mono + Data (352kbit/s)	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyData352Kbits
19	AM mono + unrelated NICAM Mono	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyMono
20	AM mono + unrelated NICAM Stereo	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyStereo
21	AM mono + unrelated NICAM Dual	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyDual
22	AM mono + unrelated NICAM Data only (704kbit/s)	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyData704Kbits
23	AM mono + unrelated NICAM Mono + Data (352kbit/s)	uhAnaAdec_SoundStandardAmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyData352Kbits
24	FM mono + NICAM Mono	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyMono
25	FM mono + NICAM Stereo	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyStereo

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Audio Signal		Sound Standard	Additional Sound Property set
Id	**Name**		
26	FM mono + NICAM Dual	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyDual
27	FM mono + NICAM Data only (704kbit/s)	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyData704Kbits
28	FM mono + NICAM Mono + Data (352kbit/s)	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyData352Kbits
29	FM mono + unrelated NICAM Mono	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyMono
30	FM mono + unrelated NICAM Stereo	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyStereo
31	FM mono + unrelated NICAM Dual	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyDual
32	FM mono + unrelated NICAM Data only (704kbit/s)	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyData704Kbits
33	FM mono + unrelated NICAM Mono + Data (352kbit/s)	uhAnaAdec_SoundStandardFmNicam	uhAnaAdec_SoundPropertyUnRelated | uhAnaAdec_SoundPropertyMono | uhAnaAdec_SoundPropertyData352Kbits

7.1.1.5.1.7 GetStableOutput

Signature

```
uhErrorCode_t GetStableOutput (
    [out] pBool pStableOutput);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether the analog audio decoder has currently stable output.

Parameters

Table 101

Name	Description
pStableOutput	Output parameter returning whether the analog audio decoder has currently stable output.

Return Values

Standard.

Precondition

True

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pStableOutput == dec.stableOutput
```

7.1.1.5.2 uhIAnaAdecNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a decoder. The `Subscribe` function of the `uhIAnaAdec` interface can be used to subscribe to these events.

Interface ID
```
uuid( DC8AA7A5-7A94-4242-8A29-5388210753F7 )
```

Execution Constraints
None.

7.1.1.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAnaAdec_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

7.1.1.5.2.2 OnAudioSignalChanged

Signature
```
[async] Void OnAudioSignalChanged (
    [in] UInt32 cookie,
    [in] uhAnaAdec_SoundStandard_t soundStandard,
    [in] uhAnaAdec_SoundPropertySet_t soundProperties );
```

Qualifiers
None.

Description
This function is called whenever a change of the audio signal, as observable by GetAudioSignal(…), is detected. This can be caused by the following events:

— loss of synchronization, either spontaneously (in particular due to bad reception conditions) or at the beginning of a (re-)tuning operation;

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— detection of the sound standard after re-synchronization;

— detection of a change in the broadcasted sound standard;

— modification of one or more of the properties of the input signal.

Parameters

Table 102

Name	Description
cookie	The value that was passed by the client at subscription time.
soundStandard	The new sound standard of the input signal as detected by the decoder.
soundProperties	Information about the additional sound properties of the input signal of the decoder besides the sound standard.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action

Postcondition
True

Remarks
Sound standard and audio program information are combined in this function, thus providing synchronous, consistent information on the input stream.

7.1.1.5.2.3 OnStableOutput

Signature

```
[async] Void OnStableOutput (
    [in] UInt32 cookie,
    [in] Bool stableOutput );
```

Qualifiers
None.

Description
The AnaAdec calls this function when there is a change in the stability of the output of the AnaAdec.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 103

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`stableOutput`	If `True`: `AnaAdec` is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream. If `False`: `AnaAdec` is no longer outputting a stable (decoded) audio stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2 UhlAtscDec

7.1.2.1 Concepts

7.1.2.1.1 Introduction

The ATSC Decoder described in this document is designed for decoding live broadcast streams as well as decoding an SPTS, which the client has for example read from a file or disc and injected into the platform (e.g. in the case of PVR). The ATSC Decoder can also decode audio and video clips. The concepts which make up the ATSC Decoder are as shown in Figure 20. These concepts are further detailed in the following subclauses.

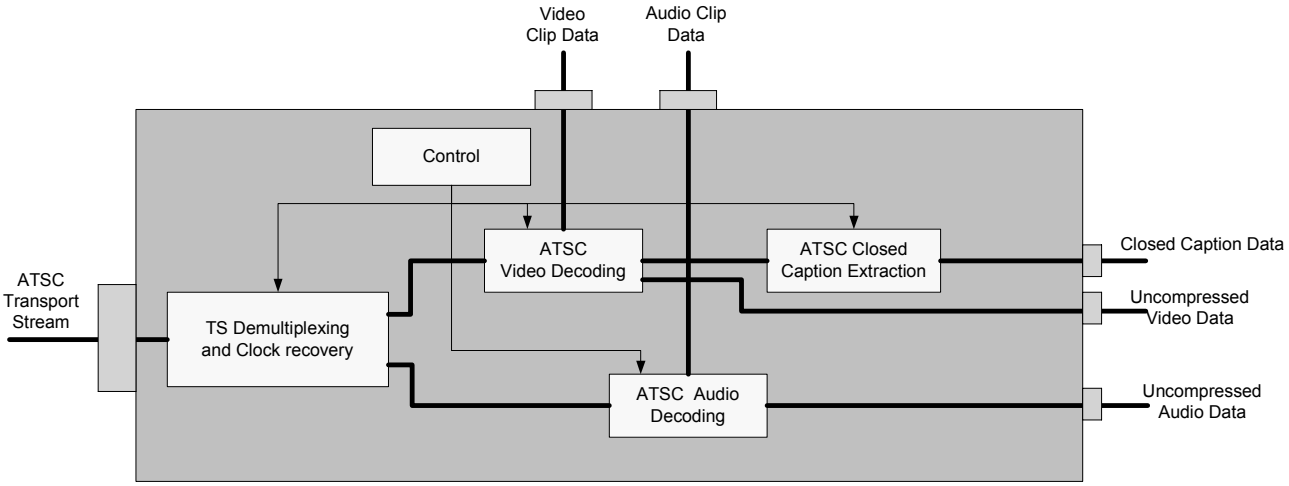


Figure 20 — ATSC Decoder concepts

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.1.2 TS Demultiplexing and Clock Management

7.1.2.1.2.1 TS Demultiplexing

The function of a TS Demultiplexer is to filter specific contents (e.g. PES packets, MPEG-2 sections) from one transport stream (TS). The client can specify several criteria for filtering. The filtered data is routed to decoders and/or (client) middleware stacks. Several types of filters exist, each with their own filter criteria. Please refer to 8.1.13 for further information.

7.1.2.1.2.2 Clock Management

The Clock management is responsible for reconstruction and maintaining the STC. In push mode (see subclause 7.1.2.1.6.1) the STC frequency and actual values in time are locked to the incoming PCR values; in pull mode (see subclause 7.1.2.1.6.2), the STC frequency is determined by the platform and the actual values are linked to the PTS values. STC is used for presentation of time synchronized program elements. Further it also provides functions to set alarms. When an alarm expires, the Clock manager sends an event to the client. Please refer to 8.1.12 for further information.

7.1.2.1.3 ATSC Video Decoding

The ATSC Video Decoder decodes an incoming ATSC video stream into an uncompressed format suitable for further processing and display. The ATSC video stream is encoded using MPEG-2 compression standard along with the constraints as specified in ATSC standard [13].

The clients, on subscription will receive a notification of the encoding standard detected along with MPEG-2 video level and MPEG-2 video profile information. Please refer to MPEG-2 video standard [2] for more details on levels and profiles.

The ATSC Video Decoder also outputs “Picture User Data” present in the ATSC video stream on a streaming output channel to facilitate extraction of Closed Caption information. For more details on Picture User Data, please refer to ATSC standard [13].

Clients on subscription can also receive “User Data” via notifications. Client can also use this “User Data” to extract Closed Caption information.

Clip play mode

In clip play mode, the ATSC Video Decoder can decode MPEG-2 video clips into a format suitable for further processing and display. For this, the client provides buffers filled with video clip data, and the ATSC Video Decoder will decode these buffers in the same order they were provided. Once it has decoded a buffer, it returns the buffer to the client.

The clients on subscription for video clip notifications will receive notifications for events such as video clip play decoding started, video clip play decoding stopped, video clip play decoding paused. The decoding of a video clip by the ATSC Video decoder is completely independent of the audio clip decoding.

Decoding in the clip play mode has priority over decoding the input coming from the demultiplexer.

Therefore, the ATSC Video decoder will stop the ATSC video stream decoding when the client requests to play the video clips and it should wait for the client buffer(s) filled with data. The client has the responsibility to give the clip data buffers in proper sequence via the `uhIAtscDecVideoClip` interface. The ATSC Video decoder will send the corresponding notification to the client(s) via `uhIAtscDecVideoClipNtf` interface, when the data contained in the buffer(s) has been decoded. When the client requests to stop the video clip play decoding, the ATSC Video decoder resumes decoding the ATSC video stream coming from the demultiplexer.

The state of the demultiplexer is unaffected by the clip play state. While clip play is started, the demultiplexer output to the ATSC video decoder is ignored.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Clip play is only allowed in the pull mode (see subclause 7.1.2.1.6).

7.1.2.1.4 ATSC Audio Decoding

The ATSC Audio Decoder decodes an incoming ATSC audio stream into an uncompressed format suitable for further processing.

The ATSC Audio Decoder supports the audio compression standard as defined in the ATSC standard, i.e. the AC-3 compression standard [5].

The Decoder has a unique output, which is completely determined by the content of its encoded audio input stream. This can be a multi-channel, a stereo or a mono audio signal. The number of output channels is identical to the number of channels in the audio input stream. Channel selection (i.e. down-mixing, up-mixing) is -not- provided by ATSC Audio Decoder.

The Decoder detects whether the digital audio input stream is encoded with an audio compression standard defined in the ATSC standard. If so, the Decoder decodes the digital audio input stream. Clients, when subscribed, receive notification of changes to the properties of the digital audio input signal.

`Clip play mode`
In the clip play mode, the ATSC Audio Decoder can decode AC-3 and MPEG-1 audio clips into a format suitable for further processing and rendering.

The ATSC Audio Decoder supports audio clip play similarly to how the ATSC Video Decoder supports video clip play (see subclause 7.1.2.1.3). The decoding of an audio clip by the ATSC Audio decoder is completely independent of the video clip decoding.

7.1.2.1.5 ATSC Closed Caption Extractor

`Input Stream`
In the case of an ATSC digital TV broadcast (terrestrially or re-transmitted over cable), closed caption data can be available in the Picture User Data bits of the video elementary stream, see Figure 21.

The input for the closed caption data-extracting unit (or closed caption extraction) is this Picture User Data.

The DTVCC Service Directory can be carried as descriptor information in the Program Mapping Table (PMT) and/or the Event Information Table (EIT). These Caption Service Descriptors, when present, identify the type of Caption Service(s) available in the bit stream. A client can access EIT and/or PMT using the TS Demultiplexer.

Two types of closed caption data can be available in the Picture User Data bits of a video elementary stream:

— Analog CC data: NTSC line 21, Field 1 and/or Field 2 closed caption data (compliant with EIA-608); see [14].

— Digital CC data (compliant with EIA-708): zero or more Caption Services are transmitted within the DTVCC Caption Channel (see Figure 21); see [15].

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

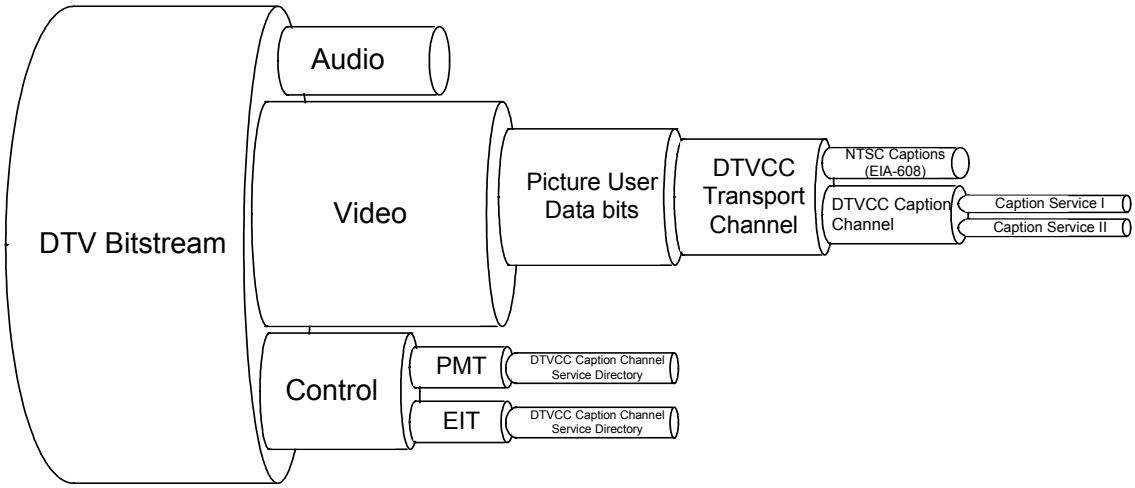


Figure 21 — Closed Caption Services and Data in an ATSC Bitstream

Closed Caption Extractor

The Closed Caption Extractor monitors the user data input stream, extracts the data payload related to closed caption, and routes it to the subscribed client(s). NTSC Caption data (analog VBI data) as well as DTVCC Caption data (in Channel Packets or in Service Blocks) can be extracted from the user data stream and routed to the subscribed client(s).

7.1.2.1.6 Push vs. pull

The ATSC decoder can operate both in the so-called push mode ("the stream is pushed to the decoders") and in the so-called pull mode ("the decoders pull the stream"). It is the responsibility of the client to set the mode such that the mode is compatible with the input stream and a setting on the TS demultiplexer, as described below.

7.1.2.1.6.1 Push mode

In push mode, the System Time Clock (STC) is locked to the PCR values of the incoming transport stream. The ATSC decoder must operate in push mode if either

— the ATSC decoder has a live broadcast transport stream input, or

the ATSC decoder has a client-provided[2] TS (or SPTS) input where each TS packet is preceded by a timestamp and the TS demultiplexer has been instructed to use these timestamps for determining when to demultiplex the packets. (NOTE The client can for instance receive a time-stamped TS (or SPTS) over a home network; see for instance the DLNA specifications [16].)

This mode is called the push mode, because the stream is "pushed" to the decoders. The stream itself determines the timing of the decoding.

7.1.2.1.6.2 Pull mode

In pull mode, the STC is not locked to the PCR values of the incoming transport stream; instead, the STC has an autonomous clock frequency and the value of the STC is linked to the PTSs in the decoded video or audio elementary stream. The ATSC decoder must operate in pull mode if either

2 The ATSC decoder itself does not provide functionality for a client to provide a TS or SPTS for decoding; another component in the platform must offer this functionality (for example, the Data Injecting logical component 8.3.1 UhlDataInject).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— the ATSC decoder has a client-provided TS (or SPTS) input where the TS packets are not preceded by a timestamp, or

— the ATSC decoder has a client-provided TS (or SPTS) input where each TS packet is preceded by a timestamp and the TS demultiplexer has been instructed *not* to use these timestamps.

The pull mode is typically used in a PVR context, where the client provides a TS (or SPTS) read from a storage device (e.g., from a hard disk).

This mode is called the pull mode, because the video or audio decoder can “pull” the stream from the source (one of them determines the timing of the decoding). The TS demultiplexer must ensure, if possible, that there is enough data available in the video and audio input buffers such that the video and audio decoders can decode the stream without an underrun. The client is responsible for providing the required streaming data in time, such that the TS demultiplexer can ensure that the decoders do not get an underrun. If the video or audio input buffer is full, then the TS demultiplexer must temporarily suspend the demultiplexing until there is again space available in both the video and audio decoder input buffers.

7.1.2.1.7 Starting and stopping

The client can start and stop the TS demultiplexer and the client can start and stop the decoding (which starts and stops the audio decoder, video decoder, STC decoder and CC extractor together).

When the TS demultiplexer is stopped, its own input buffer is flushed and the input buffers of the video and audio decoders are flushed. (As the CC extractor receives its input from the same input buffer, also its input is flushed.)

Note that stopping and starting the TS demultiplexer is useful in, for example, the following situations:

— The client is playing a stream in pull mode and wants to jump to another part of the stream (e.g. 10 minutes back in the stream). Stopping the TS demultiplexer flushes the input buffers of the TS demultiplexer, the audio decoder and the video decoder. When the client then starts playing from the new location in the stream, the TS demultiplexer and the audio and video decoder can process the new stream directly, as there is no data from the ‘old’ location in the stream present anymore. This implies that any information the client obtains about the state of the TS demultiplexer, after starting it again, will always relate the new stream.

— The client wants to switch from a stream that does not have TS packet timestamps to a stream that has these timestamps (or vice versa). The client first stops the TS demultiplexer before it sets the appropriate “timestamp mode” on the TS demultiplexer.

It depends on whether the ATSC decoder operates in the push or the pull mode what happens with the streaming output of the TS demultiplexer when the decoding is in the stopped state.

In push mode, the streaming output of the TS demultiplexer is “thrown away” (= ignored; switches are open) until the decoding is started again (see Figure 22). This prevents overflow of the input buffers. Apart from that its streaming output is “thrown away”, the TS demultiplexer continues to function normally.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

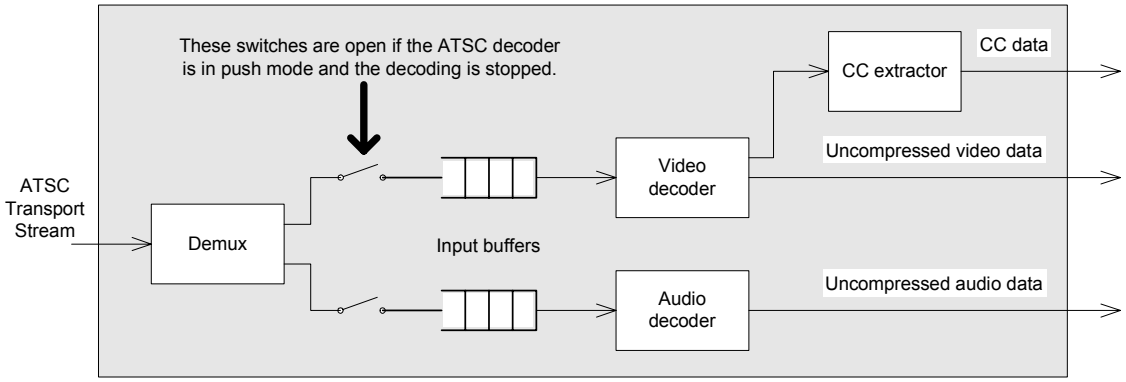


Figure 22 — Routing of the streaming output of the TS demultiplexer when decoding is in the stopped state.

In pull mode, the streaming output of the TS demultiplexer continues to be streamed into the input buffers of the video and audio decoders (i.e., if and when the demux is started), until the buffers are full; then the demultiplexing is temporarily suspended until there is again space available in both the video and audio decoder input buffers (i.e., after the decoders have been started again).

7.1.2.1.8 Trick modes

When the ATSC decoder operates in pull mode, it can support a number of trick modes. Examples of such trick modes are:

— Playback at a different speed (e.g. at half speed or double speed)

— Pause

— Step to the next frame when paused

— Decoding of only I-frames, or only I- and P-frames

7.1.2.2 Types & Constants

7.1.2.2.1 Public Types & Constants

7.1.2.2.1.1 Error Codes

Signature

```
const uhErrorCode_t  UH_ERR_ATSCDEC_NO_CCEXTR            = 0x00000001;
const uhErrorCode_t  UH_ERR_ATSCDEC_PAUSE_AT_NOT_ENABLED = 0x00000002;
const uhErrorCode_t  UH_ERR_ATSCDEC_CANCEL_IGNORED       = 0x00000003;
const uhErrorCode_t  UH_ERR_ATSCDEC_STOP_IGNORED         = 0x00000004;
const uhErrorCode_t  UH_ERR_ATSCDEC_PAUSE_IGNORED        = 0x00000005;
```

Qualifiers

— error-codes

Description

Defines all non-standard errors that may be returned by the interfaces of AtscDec.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Constants

Table 104

Name	Description
UH_ERR_ATSCDEC_NO_CCEXTR	This AtscDec does not support AtscDecCcExtr functionality.
UH_ERR_ATSCDEC_PAUSE_AT_NOT_ENABLED	There is no “Pause At” request enabled at this moment.
UH_ERR_ATSCDEC_CANCEL_IGNORED	The request to cancel has been ignored.
UH_ERR_ATSCDEC_STOP_IGNORED	The request to stop has been ignored.
UH_ERR_ATSCDEC_PAUSE_IGNORED	The request to pause has been ignored.

7.1.2.2.1.2 uhAtscDec_ValStat_t

Signature

```
typedef enum _uhAtscDec_ValStat_t {
    uhAtscDec_ValStatInvalid = -2,
    uhAtscDec_ValStatKnown   =  1
} uhAtscDec_ValStat_t, *puhAtscDec_ValStat_t;
```

Qualifiers

None.

Description

This type represents the validity status of the related property/properties.

Values

Table 105

Name	Description
uhAtscDec_ValStatInvalid	The property/properties are temporarily not available.
uhAtscDec_ValStatKnown	The property/properties are available and determined.

7.1.2.2.1.3 uhAtscDecCtrl_StartStopNtf_t

Signature

```
typedef enum _uhAtscDecCtrl_StartStopNtf_t {
    uhAtscDecCtrl_StartStopSubscriptionChanged = 0x00000001,
    uhAtscDecCtrl_StartStopDemuxStarted        = 0x00000002,
    uhAtscDecCtrl_StartStopDemuxStoppped       = 0x00000004,
    uhAtscDecCtrl_StartStopDecodersStarted     = 0x00000008,
    uhAtscDecCtrl_StartStopDecodersStopped     = 0x00000010
} uhAtscDecCtrl_StartStopNtf_t, *puhAtscDecCtrl_StartStopNtf_t;
```

Qualifiers

— enum-element

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAtscDecStartStopNtf`. These values can be used to individually turn on and off the notifications.

Values

Table 106

Name	Description
`uhAtscDecCtrl_StartStopSubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhAtscDecCtrl_StartStopDemuxStarted`	Represents `OnDemuxStarted`.
`uhAtscDecCtrl_StartStopDemuxStoppped`	Represents `OnDemuxStopped`.
`uhAtscDecCtrl_StartStopDecodersStarted`	Represents `OnDecodersStarted`.
`uhAtscDecCtrl_StartStopDecodersStopped`	Represents `OnDecodersStopped`.

7.1.2.2.1.4 uhAtscDecCtrl_StartStopNtfSet_t

Signature
```
typedef Int32 uhAtscDecCtrl_StartStopNtfSet_t, *puhAtscDecCtrl_StartStopNtfSet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions, which are typically constructed by logical OR-ing of values of type `uhAtscDecCtrl_StartStopNtf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

7.1.2.2.1.5 uhAtscDecCtrl_StartStopAllNtfs

Signature
```
const uhAtscDecCtrl_StartStopNtfSet_t uhAtscDecCtrl_StartStopAllNtfs = 0x1F;
```

Qualifiers
None.

Description
Defines the set of all values of type `uhAtscDecCtrl_StartStopNtf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.6 uhAtscDecCtrl_TrickNtf_t

Signature

```
typedef enum _uhAtscDecCtrl_TrickNtf_t {
    uhAtscDecCtrl_TrickSubscriptionChanged = 0x00000001,
    uhAtscDecCtrl_TrickPaused              = 0x00000002,
    uhAtscDecCtrl_TrickContinued           = 0x00000004,
    uhAtscDecCtrl_TrickStepped             = 0x00000008,
    uhAtscDecCtrl_TrickNextIframeDecoded   = 0x00000010
} uhAtscDecCtrl_TrickNtf_t, *puhAtscDecCtrl_TrickNtf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAtscDecTrickNtf`. These values can be used to individually turn on and off the notifications.

Values

Table 107

Name	Description
`uhAtscDecCtrl_TrickSubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhAtscDecCtrl_TrickPaused`	Represents `OnPaused`.
`uhAtscDecCtrl_TrickContinued`	Represents `OnContinued`.
`uhAtscDecCtrl_TrickStepped`	Represents `OnStepped`.
`uhAtscDecCtrl_TrickNextIframeDecoded`	Represents `OnNextIframeDecoded`.

7.1.2.2.1.7 uhAtscDecCtrl_TrickNtfSet_t

Signature

```
typedef Int32 uhAtscDecCtrl_TrickNtfSet_t, *puhAtscDecCtrl_TrickNtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions, which are typically constructed by logical OR-ing of values of type `uhAtscDecCtrl_TrickNtf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

7.1.2.2.1.8 uhAtscDecCtrl_TrickAllNtfs

Signature

```
const uhAtscDecCtrl_TrickNtfSet_t uhAtscDecCtrl_TrickAllNtfs = 0x1F;
```

Qualifiers

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set of all values of type `uhAtscDecCtrl_TrickNtf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

7.1.2.2.1.9 uhAtscDecCtrl_State_t

Signature
```
typedef enum _uhAtscDecCtrl_State_t {
    uhAtscDecCtrl_StateStopped  = 0,
    uhAtscDecCtrl_StateStarting = 1,
    uhAtscDecCtrl_StateStarted  = 2,
    uhAtscDecCtrl_StateStopping = 3
} uhAtscDecCtrl_State_t, *puhAtscDecCtrl_State_t;
```

Qualifiers
None.

Description
Defines the operational states of the de-multiplexer and the decoders.

Values

Table 108

Name	Description
`uhAtscDecCtrl_StateStopped`	Represents the Stopped state.
`uhAtscDecCtrl_StateStarting`	Represents the Starting state.
`uhAtscDecCtrl_StateStarted`	Represents the Started state.
`uhAtscDecCtrl_StateStopping`	Represents the Stopping state.

7.1.2.2.1.10 uhAtscDec_FrameType_t

Signature
```
typedef enum _uhAtscDec_FrameType_t {
    uhAtscDec_Iframe    = 0x00000001,
    uhAtscDec_IorPframe = 0x00000002,
    uhAtscDec_AnyFrame  = 0x00000004
} uhAtscDec_FrameType_t, *puhAtscDec_FrameType_t;
```

Qualifiers
— enum-element

Description
This type defines frame types.

Values

Table 109

Name	Description
`uhAtscDec_Iframe`	I-frame.
`uhAtscDec_IorPframe`	I-frame or P-frame.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhAtscDec_AnyFrame	Any frame (I, P or B).

7.1.2.2.1.11 uhAtscDec_FrameTypeSet_t

Signature

```
typedef Int32 uhAtscDec_FrameTypeSet_t, *puhAtscDec_FrameTypeSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of frame types.

Constraints

None.

7.1.2.2.1.12 uhAtscDec_TrickState_t

Signature

```
typedef enum _uhAtscDec_TrickState_t {
    uhAtscDec_TrickStateStopped         = 0,
    uhAtscDec_TrickStateStarting        = 1,
    uhAtscDec_TrickStateStarted         = 2,
    uhAtscDec_TrickStateStopping        = 3,
    uhAtscDec_TrickStatePaused          = 4,
    uhAtscDec_TrickStatePausing         = 5,
    uhAtscDec_TrickStateContinuing      = 6,
    uhAtscDec_TrickStateStepping        = 7,
    uhAtscDec_TrickStateDecIframe       = 8
} uhAtscDec_TrickState_t, *puhAtscDec_TrickState_t;
```

Qualifiers

None.

Description

Defines the decoding trick states.

Values

Table 110

Name	Description
uhAtscDec_TrickStateStopped	The decoders are stopped.
uhAtscDec_TrickStateStarting	The decoders are in transition from the stopped state to the started state.
uhAtscDec_TrickStateStarted	The decoders are started.
uhAtscDec_TrickStateStopping	The decoders are in transition to the stopped state.
uhAtscDec_TrickStatePaused	The decoders are paused.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhAtscDec_TrickStatePausing	The decoders are in transition to the paused state.
uhAtscDec_TrickStateContinuing	The decoders are in transition from the paused state to the started state.
uhAtscDec_TrickStateStepping	The decoders are decoding the next frame and are in the transition back to the paused state. NOTE When stepping, no audio is being decoded; CC extraction is performed.
uhAtscDec_TrickStateDecIframe	The decoders are decoding the next I-frame and are in the transition back to the stopped state. NOTE When decoding the next I-frame, no audio is being decoded; CC extraction is performed.

7.1.2.2.1.13 uhAtscDec_StreamMode_t

Signature

```
typedef enum _uhAtscDec_StreamMode_t {
    uhAtscDec_StreamModePush = 0,
    uhAtscDec_StreamModePull = 1
} uhAtscDec_StreamMode_t, *puhAtscDec_StreamMode_t;
```

Qualifiers
None.

Description
Defines the operational stream modes of the ATSC decoder.

Values

Table 111

Name	Description
uhAtscDec_StreamModePush	Represents the push mode. See subclause 7.1.2.1.6 for more details.
uhAtscDec_StreamModePull	Represents the pull mode. See subclause 7.1.2.1.6 for more details.

7.1.2.2.1.14 uhAtscDec_Speed_t

Signature

```
typedef struct _uhAtscDec_Speed_t {
    UInt32 numerator;
    UInt32 denominator;
} uhAtscDec_Speed_t, *puhAtscDec_Speed_t;
```

Qualifiers
None.

Description
Values of this type represent distinct values of speed, relative to normal speed. Speed is expressed in numerator / denominator. The numerator and denominator have the smallest possible value for which the expression `speed == numerator / denominator` still holds (e.g. `numerator == denominator == 1` for normal speed).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Fields

Table 112

Name	Description
numerator	The numerator of the speed.
denominator	The denominator of the speed.

Constraints

```
numerator != 0

denominator != 0
```

7.1.2.2.1.15 uhAtscDecVideo_Ntf_t

Signature

```
typedef enum _uhAtscDecVideo_Ntf_t {
    uhAtscDecVideo_SubscriptionChanged = 0x00000001,
    uhAtscDecVideo_StandardDataChanged = 0x00000002,
    uhAtscDecVideo_PictureDataChanged  = 0x00000004,
    uhAtscDecVideo_Error               = 0x00000008,
    uhAtscDecVideo_Recovered           = 0x00000010,
    uhAtscDecVideo_StableOutput        = 0x00000020
} uhAtscDecVideo_Ntf_t, *puhAtscDecVideo_Ntf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAtscDecVideoNtf`. These values can be used to individually turn on and off the notifications.

Values

Table 113

Name	Description
uhAtscDecVideo_SubscriptionChanged	Represents `OnSubscriptionChanged`
uhAtscDecVideo_StandardDataChanged	Represents `OnStandardDataChanged`
uhAtscDecVideo_PictureDataChanged	Represents `OnPictureDataChanged`
uhAtscDecVideo_Error	Represents `OnError`
uhAtscDecVideo_Recovered	Represents `OnRecovered`
uhAtscDecVideo_StableOutput	Represents `OnStableOutput`

--`,,```,,,,``-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.16 uhAtscDecVideo_NtfSet_t

Signature

```
typedef Int32 uhAtscDecVideo_NtfSet_t, *puhAtscDecVideo_NtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions, which are typically constructed by logical OR-ing of values of type uh`AtscDecVideo_Ntf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

7.1.2.2.1.17 uhAtscDecVideo_AllNtfs

Signature

```
const uhAtscDecVideo_NtfSet_t uhAtscDecVideo_AllNtfs = 0x0000003F;
```

Qualifiers

None.

Description

Defines the set of all values of type `uhAtscDecVideo_Ntf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

7.1.2.2.1.18 uhAtscDecVideo_UserDataNtf_t

Signature

```
typedef enum uhAtscDecVideo_UserDataNtf_t {
    uhAtscDecVideo_UserDataSubscriptionChanged = 0x00000001,
    uhAtscDecVideo_UserDataPicture             = 0x00000002,
    uhAtscDecVideo_UserDataGop                 = 0x00000004,
    uhAtscDecVideo_UserDataSequence            = 0x00000008
} uhAtscDecVideo_UserDataNtf_t, *puhAtscDecVideo_UserDataNtf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. There is a one-to-one correspondence between these values and the functions in the notification interface `uhIAtscDecVideoUserDataNtf`. These values can be used to individually turn on and off notifications.

Values

Table 114

Name	Description
`uhAtscDecVideo_UserDataSubscriptionChanged`	Represents `OnUserDataSubscriptionChanged`.
`uhAtscDecVideo_UserDataPicture`	Represents `OnUserDataPictureArrived`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhAtscDecVideo_UserDataGop	Represents OnUserDataGopArrived
uhAtscDecVideo_UserDataSequence	Represents OnUserDataSequenceArrived

7.1.2.2.1.19 uhAtscDecVideo_UserDataNtfSet_t

Signature

```
typedef Int32 uhAtscDecVideo_UserDataNtfSet_t, *puhAtscDecVideo_UserDataNtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions, which are typically constructed by logical OR-ing of values of type uhAtscDecVideo_UserDataNtf_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

7.1.2.2.1.20 uhAtscDecVideo_UserDataAllNtfs

Signature

```
const uhAtscDecVideo_UserDataNtfSet_t uhAtscDecVideo_UserDataAllNtfs = 0x0000000F;
```

Qualifiers

None.

Description

Defines the set of all values of type uhAtscDecVideo_UserDataNtf_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

7.1.2.2.1.21 uhAtscDecVideo_ClipState_t

Signature

```
typedef enum _uhAtscDecVideo_ClipState_t {
    uhAtscDecVideo_ClipStStarted   = 0,
    uhAtscDecVideo_ClipStStarting  = 1,
    uhAtscDecVideo_ClipStStopped   = 2,
    uhAtscDecVideo_ClipStStopping  = 3,
    uhAtscDecVideo_ClipStPaused    = 4,
    uhAtscDecVideo_ClipStPausing   = 5
} uhAtscDecVideo_ClipState_t, *puhAtscDecVideo_ClipState_t;
```

Qualifiers

None.

Description

Values of this type represent the various states of the ATSC Video Decoder for Clip Play.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 115

Name	Description
uhAtscDecVideo_ClipStStarted	Represents Video Clip Play is started.
uhAtscDecVideo_ClipStStarting	Represents Video Clip Play is going to start.
uhAtscDecVideo_ClipStStopped	Represents Video ClipPlay is stopped.
uhAtscDecVideo_ClipStStopping	Represents Video Clip Play is going to stop.
uhAtscDecVideo_ClipStPaused	Represents Video Clip Play is paused.
uhAtscDecVideo_ClipStPausing	Represents Video Clip Play is going to pause.

7.1.2.2.1.22 uhAtscDecVideo_ClipNtf_t

Signature

```
typedef enum _uhAtscDecVideo_ClipNtf_t {
    uhAtscDecVideo_ClipSubscriptionChanged = 0x00000001,
    uhAtscDecVideo_ClipBufferAvailable     = 0x00000002,
    uhAtscDecVideo_ClipEndOfData           = 0x00000004,
    uhAtscDecVideo_ClipStarted             = 0x00000008,
    uhAtscDecVideo_ClipStopped             = 0x00000010,
    uhAtscDecVideo_ClipPaused              = 0x00000020
} uhAtscDecVideo_ClipNtf_t, *puhAtscDecVideo_ClipNtf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAtscDecVideoClipNtf`. The values can be used to individually turn on and off the notifications.

Values

Table 116

Name	Description
uhAtscDecVideo_ClipSubscriptionChanged	Represents `OnSubscriptionChanged`.
uhAtscDecVideo_ClipBufferAvailable	Represents `OnBufferAvailable`
uhAtscDecVideo_ClipEndOfData	Represents `OnEndOfData`
uhAtscDecVideo_ClipStarted	Represents `OnStarted`
uhAtscDecVideo_ClipStopped	Represents `OnStopped`
uhAtscDecVideo_ClipPaused	Represents `OnPaused`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.23 uhAtscDecVideo_ClipNtfSet_t

Signature

```
typedef Int32 uhAtscDecVideo_ClipNtfSet_t, *puhAtscDecVideo_ClipNtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhAtscDec`Video_ClipNtf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

7.1.2.2.1.24 uhAtscDecVideo_ClipAllNtfs

Signature

```
const uhAtscDecVideo_ClipNtfSet_t uhAtscDecVideo_ClipAllNtfs = 0x0000003F;
```

Qualifiers

None.

Description

Defines the set of all values of type uhAtscDec`Video_ClipNtf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

7.1.2.2.1.25 uhAtscDecVideo_CompStd_t

Signature

```
typedef enum _uhAtscDecVideo_CompStd_t {
    uhAtscDecVideo_CompStdInvalid = -2,
    uhAtscDecVideo_CompStdUnknown = -1,
    uhAtscDecVideo_CompStdMpeg2   =  1
} uhAtscDecVideo_CompStd_t, *puhAtscDecVideo_CompStd_t;
```

Qualifiers

None.

Description

Values of this type represent digital video compression standards.

— The MPEG-2 Standard is defined in [2].

Values

Table 117

Name	Description
`uhAtscDecVideo_CompStdInvalid`	The compression standard is temporarily not available.
`uhAtscDecVideo_CompStdUnknown`	The compression standard is available, yet could not be determined.
`uhAtscDecVideo_CompStdMpeg2`	The compression standard of the input stream is MPEG-2.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.26 uhAtscDecVideo_Level_t

Signature

```
typedef enum _uhAtscDecVideo_Level_t {
    uhAtscDecVideo_LevelInvalid  = -2,
    uhAtscDecVideo_LevelUnknown  = -1,
    uhAtscDecVideo_LevelLow      =  1,
    uhAtscDecVideo_LevelMain     =  2,
    uhAtscDecVideo_LevelHigh1440 =  3,
    uhAtscDecVideo_LevelHigh     =  4
} uhAtscDecVideo_Level_t, *puhAtscDecVideo_Level_t;
```

Qualifiers
None.

Description
Values of this type represent various MPEG-2 video levels.

Values

Table 118

Name	Description
uhAtscDecVideo_LevelInvalid	The video level is temporarily not available.
uhAtscDecVideo_LevelUnknown	The video level is available, yet could not be determined.
uhAtscDecVideo_LevelLow	Low level
uhAtscDecVideo_LevelMain	Main level
uhAtscDecVideo_LevelHigh1440	High-1440 level
uhAtscDecVideo_LevelHigh	High level

7.1.2.2.1.27 uhAtscDecVideo_Profile_t

Signature

```
typedef enum _uhAtscDecVideo_Profile_t {
    uhAtscDecVideo_InvalidProfile     = -2,
    uhAtscDecVideo_UnknownProfile     = -1,
    uhAtscDecVideo_SimpleProfile      =  1,
    uhAtscDecVideo_MainProfile        =  2,
    uhAtscDecVideo_SnrScalableProfile =  3,
    uhAtscDecVideo_SpatScaleProfile   =  4,
    uhAtscDecVideo_HighProfile        =  5
} uhAtscDecVideo_Profile_t, *puhAtscDecVideo_Profile_t;
```

Qualifiers
None.

Description
Values of this type represent the various MPEG-2 profiles.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 119

Name	Description
uhAtscDecVideo_InvalidProfile	The MPEG-2 profile is temporarily not available.
uhAtscDecVideo_UnknownProfile	The MPEG-2 profile is available, yet could not be determined.
uhAtscDecVideo_SimpleProfile	Simple Profile
uhAtscDecVideo_MainProfile	Main Profile
uhAtscDecVideo_SnrScalableProfile	SNR Scalable Profile
uhAtscDecVideo_SpatScaleProfile	Spatially Scalable Profile
uhAtscDecVideo_HighProfile	High Profile

7.1.2.2.1.28 uhAtscDecVideo_FrameRate_t

Signature

```
typedef enum _uhAtscDecVideo_FrameRate_t {
    uhAtscDecVideo_InvalidFrameRate = -2,
    uhAtscDecVideo_UnknownFrameRate = -1,
    uhAtscDecVideo_23_976Hz         =  1,
    uhAtscDecVideo_24Hz             =  2,
    uhAtscDecVideo_29_97Hz          =  3,
    uhAtscDecVideo_30Hz             =  4,
    uhAtscDecVideo_59_94Hz          =  5,
    uhAtscDecVideo_60Hz             =  6
} uhAtscDecVideo_FrameRate_t, *puhAtscDecVideo_FrameRate_t;
```

Qualifiers

None.

Description

Values of this type represent frame rates of the incoming video signal.

Values

Table 120

Name	Description
uhAtscDecVideo_InvalidFrameRate	The frame rate is temporarily not available.
uhAtscDecVideo_UnknownFrameRate	The frame rate is available, yet could not be determined.
uhAtscDecVideo_23_976Hz	23.976 frames per second.
uhAtscDecVideo_24Hz	24 frames per second.
uhAtscDecVideo_29_97Hz	29.97 frames per second.
uhAtscDecVideo_30Hz	30 frames per second.
uhAtscDecVideo_59_94Hz	59.94 frames per second.
uhAtscDecVideo_60Hz	60 frames per second.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.29 uhAtscDecVideo_ScanType_t

Signature

```
typedef enum _uhAtscDecVideo_ScanType_t {
    uhAtscDecVideo_ScanTypeInvalid     = -2,
    uhAtscDecVideo_ScanTypeUnknown     = -1,
    uhAtscDecVideo_ScanTypeInterlaced  =  1,
    uhAtscDecVideo_ScanTypeProgressive =  2
} uhAtscDecVideo_ScanType_t, *puhAtscDecVideo_ScanType_t;
```

Qualifiers
None.

Description
Values of this type represent scan types.

Values

Table 121

Name	Description
uhAtscDecVideo_ScanTypeInvalid	The scan type is temporarily not available.
uhAtscDecVideo_ScanTypeUnknown	The scan type is available, yet could not be determined.
uhAtscDecVideo_ScanTypeInterlaced	Interlaced scanning.
uhAtscDecVideo_ScanTypeProgressive	Progressive scanning.

7.1.2.2.1.30 uhAtscDecVideo_Error_t

Signature

```
typedef enum _uhAtscDecVideo_Error_t {
    uhAtscDecVideo_NoError       = 0x00000000,
    uhAtscDecVideo_DecodePicture = 0x00000001,
    uhAtscDecVideo_Underrun      = 0x00000002
} uhAtscDecVideo_Error_t, *puhAtscDecVideo_Error_t;
```

Qualifiers
— enum-element

Description
This type enumerates the errors that can occur in the video decoder.

Values

Table 122

Name	Description
uhAtscDecVideo_NoError	Indicates that there is no error.
uhAtscDecVideo_DecodePicture	Indicates an error occurred while decoding a picture of video
uhAtscDecVideo_Underrun	Indicates an underrun error due to insufficient data

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.31 uhAtscDecVideo_ErrorSet_t

Signature

```
typedef Int32 uhAtscDecVideo_ErrorSet_t, *puhAtscDecVideo_ErrorSet_t;
```

Qualifiers

— enum-set

Description

This type represents sets of errors that can occur in the video decoder. Values of this type are typically constructed by logical OR-ing of values of type uh`AtscDecVideo_Error_t`.

7.1.2.2.1.32 uhAtscDecAudio_CompStd_t

Signature

```
typedef enum _uhAtscDecAudio_CompStd_t {
    uhAtscDecAudio_CompStdInvalid = -2,
    uhAtscDecAudio_CompStdUnknown = -1,
    uhAtscDecAudio_CompStdAc3     =  1,
    uhAtscDecAudio_Mpeg1Layer1    =  2,
    uhAtscDecAudio_Mpeg1Layer2    =  3,
    uhAtscDecAudio_Mpeg1Layer3    =  4
} uhAtscDecAudio_CompStd_t, *puhAtscDecAudio_CompStd_t;
```

Qualifiers

None.

Description

Values of this type represent digital audio compression standards.

— The AC-3 Standard is defined in [5]. See [13], for restriction of the AC-3 standard for ATSC.

— The MPEG-1 Standards are defined in [1]. The MPEG-1 Standards are only used for clip play.

Values

Table 123

Name	Description
`uhAtscDecAudio_CompStdInvalid`	The compression standard is temporarily not available.
`uhAtscDecAudio_CompStdUnknown`	The compression standard is available, yet could not be determined.
`uhAtscDecAudio_CompStdAc3`	AC-3.
`uhAtscDecAudio_Mpeg1Layer1`	MPEG-1, layer I.
`uhAtscDecAudio_Mpeg1Layer2`	MPEG-1, layer II.
`uhAtscDecAudio_Mpeg1Layer3`	MPEG-1, layer III.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.33 uhAtscDecAudio_SampFreq_t

Signature

```
typedef enum _uhAtscDecAudio_SampFreq_t {
    uhAtscDecAudio_FreqInvalid = -2,
    uhAtscDecAudio_FreqUnknown = -1,
    uhAtscDecAudio_48k         =  1,
    uhAtscDecAudio_32k         =  2,
    uhAtscDecAudio_44k1        =  3
} uhAtscDecAudio_SampFreq_t, *puhAtscDecAudio_SampFreq_t;
```

Qualifiers
None.

Description
Values of this type represent the sampling frequency of the incoming digital audio signal.

Values

Table 124

Name	Description
uhAtscDecAudio_FreqInvalid	The sampling frequency is temporarily not available.
uhAtscDecAudio_FreqUnknown	The sampling frequency is available, yet could not be determined.
uhAtscDecAudio_48k	48 kHz sampling frequency.
uhAtscDecAudio_32k	32 kHz sampling frequency.
uhAtscDecAudio_44k1	44.1 kHz sampling frequency.

7.1.2.2.1.34 uhAtscDecAudio_ServiceType_t

Signature

```
typedef enum _uhAtscDecAudio_ServiceType_t {
    uhAtscDecAudio_ServiceInvalid   = -2,
    uhAtscDecAudio_ServiceUnknown   = -1,
    uhAtscDecAudio_CompleteMain     =  1,
    uhAtscDecAudio_MusicAndEffects  =  2,
    uhAtscDecAudio_VisuallyImpaired =  3,
    uhAtscDecAudio_HearingImpaired  =  4,
    uhAtscDecAudio_Dialogue         =  5,
    uhAtscDecAudio_Commentary       =  6,
    uhAtscDecAudio_Emergency        =  7,
    uhAtscDecAudio_VoiceOver        =  8
} uhAtscDecAudio_ServiceType_t, *puhAtscDecAudio_ServiceType_t;
```

Qualifiers
None.

Description
Values of this type provide information about the type of audio service the digital audio input stream of the Decoder conveys.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 125

Name	Description
uhAtscDecAudio_ServiceInvalid	The Audio Service is temporarily not available.
uhAtscDecAudio_ServiceUnknown	The Audio Service is available, yet could not be determined.
uhAtscDecAudio_CompleteMain	Main Audio Service: Complete Main.
uhAtscDecAudio_MusicAndEffects	Main Audio Service: Music and Effects.
uhAtscDecAudio_VisuallyImpaired	Associated Audio Service: Visually Impaired.
uhAtscDecAudio_HearingImpaired	Associated Audio Service: Hearing Impaired.
uhAtscDecAudio_Dialogue	Associated Audio Service: Dialogue.
uhAtscDecAudio_Commentary	Associated Audio Service: Commentary.
uhAtscDecAudio_Emergency	Associated Audio Service: Emergency.
uhAtscDecAudio_VoiceOver	Associated Audio Service: Voice-Over.

7.1.2.2.1.35 uhAtscDecAudio_DolbySurroundInfo_t

Signature

```
typedef enum _uhAtscDecAudio_DolbySurroundInfo_t {
    uhAtscDecAudio_DolbyInvalid     = -2,
    uhAtscDecAudio_DolbyUnknown     = -1,
    uhAtscDecAudio_NotDolbySurround =  1,
    uhAtscDecAudio_DolbySurround    =  2
} uhAtscDecAudio_DolbySurroundInfo_t, *puhAtscDecAudio_DolbySurroundInfo_t;
```

Qualifiers

None.

Description

Values of this type provide information about the presence of Dolby Surround in the digital audio input stream of the Decoder.

Values

Table 126

Name	Description
uhAtscDecAudio_DolbyInvalid	The Dolby Surround info is temporarily not available.
uhAtscDecAudio_DolbyUnknown	The presence of Dolby Surround info could not be determined.
uhAtscDecAudio_NotDolbySurround	Dolby Surround is not present in the audio input stream.
uhAtscDecAudio_DolbySurround	Dolby Surround is present in the audio input stream.

© ISO/IEC 2007 – All rights reserved
--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.36 uhAtscDecAudio_Error_t

Signature

```
typedef enum _uhAtscDecAudio_Error_t {
    uhAtscDecAudio_NoError         = 0x00000000,
    uhAtscDecAudio_SyncNotFound    = 0x00000001,
    uhAtscDecAudio_InvalidHeader   = 0x00000002,
    uhAtscDecAudio_DecodeFrame     = 0x00000004,
    uhAtscDecAudio_CrcFailed       = 0x00000008,
    uhAtscDecAudio_Underrun        = 0x00000010
} uhAtscDecAudio_Error_t, *puhAtscDecAudio_Error_t;
```

Qualifiers

— enum-element

Description

This type enumerates the errors that can occur in the audio decoder.

Values

Table 127

Name	Description
uhAtscDecAudio_NoError	Indicates that there is no error.
uhAtscDecAudio_SyncNotFound	The decoder did not find a valid sync word within the search range. To maintain a continuous data stream at the output the decoder may mute one frame.
uhAtscDecAudio_InvalidHeader	Invalid header. Error while parsing the bit stream.
uhAtscDecAudio_DecodeFrame	An error occurred while decoding. The decoder may perform muting of the current audio block, and the rest of the frame.
uhAtscDecAudio_CrcFailed	The calculation of the CRC derived a wrong value. Audio block muting or audio block repeating may be performed.
uhAtscDecAudio_Underrun	The decoder requested data, but none was available.

7.1.2.2.1.37 uhAtscDecAudio_ErrorSet_t

Signature

```
typedef Int32 uhAtscDecAudio_ErrorSet_t, *puhAtscDecAudio_ErrorSet_t;
```

Qualifiers

— enum-set

Description

This type represents sets of errors that can occur in the audio decoder. Values of this type are typically constructed by logical OR-ing of values of type uh`AtscDecAudio_Error_t`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.38 uhAtscDecAudio_Ntf_t

Signature

```
typedef enum _uhAtscDecAudio_Ntf_t {
    uhAtscDecAudio_SubscriptionChanged  = 0x00000001,
    uhAtscDecAudio_StandardChanged      = 0x00000002,
    uhAtscDecAudio_SampFreqChanged      = 0x00000004,
    uhAtscDecAudio_BitRateChanged       = 0x00000008,
    uhAtscDecAudio_ChannelsChanged      = 0x00000010,
    uhAtscDecAudio_ServiceTypeChanged   = 0x00000020,
    uhAtscDecAudio_DolbySurroundChanged = 0x00000040,
    uhAtscDecAudio_Error                = 0x00000080,
    uhAtscDecAudio_Recovered            = 0x00000100,
    uhAtscDecAudio_StableOutput         = 0x00000200
} uhAtscDecAudio_Ntf_t, *puhAtscDecAudio_Ntf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAtscDecAudioNtf`. The values can be used to individually turn on and off the notifications.

Values

Table 128

Name	Description
`uhAtscDecAudio_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhAtscDecAudio_StandardChanged`	Represents `OnStandardChanged`.
`uhAtscDecAudio_SampFreqChanged`	Represents `OnSampFreqChanged`.
`uhAtscDecAudio_BitRateChanged`	Represents `OnBitRateChanged`.
`uhAtscDecAudio_ChannelsChanged`	Represents `OnChannelsChanged`.
`uhAtscDecAudio_ServiceTypeChanged`	Represents `OnServiceTypeChanged`.
`uhAtscDecAudio_DolbySurroundChanged`	Represents `OnDolbySurroundChanged`.
`uhAtscDecAudio_Error`	Some error(s) occurred while decoding the incoming signal. The exact error(s) that occurred will be an argument of the OnError notification that is generated (see type uhAtscDecAudio_Error_t).
`uhAtscDecAudio_Recovered`	Represents `OnRecovered`.
`uhAtscDecAudio_StableOuput`	Represents `OnStableOutput`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.39 uhAtscDecAudio_NtfSet_t

Signature

```
typedef Int32 uhAtscDecAudio_NtfSet_t, *puhAtscDecAudio_NtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uh`AtscDecAudio_Ntf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints

None.

7.1.2.2.1.40 uhAtscDecAudio_AllNtfs

Signature

```
const uhAtscDecAudio_Ntf_t uhAtscDecAudio_AllNtfs = 0x000003FF;
```

Qualifiers

None.

Description

Defines the set of all values of type uh`AtscDecAudio_Ntf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

7.1.2.2.1.41 uhAtscDecAudio_ClipState_t

Signature

```
typedef enum _uhAtscDecAudio_ClipState_t {
    uhAtscDecAudio_ClipStStarted   = 0,
    uhAtscDecAudio_ClipStStarting  = 1,
    uhAtscDecAudio_ClipStStopped   = 2,
    uhAtscDecAudio_ClipStStopping  = 3,
    uhAtscDecAudio_ClipStPaused    = 4,
    uhAtscDecAudio_ClipStPausing   = 5
} uhAtscDecAudio_ClipState_t, *puhAtscDecAudio_ClipState_t;
```

Qualifiers

None.

Description

Values of this type represent the various state of the ATSC Audio Decoder for Clip Play.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 129

Name	Description
uhAtscDecAudio_ClipStStarted	Represents Audio Clip Play is started.
uhAtscDecAudio_ClipStStarting	Represents Audio Clip Play is going to start.
uhAtscDecAudio_ClipStStopped	Represents Audio Clip Play is stopped.
uhAtscDecAudio_ClipStStopping	Represents Audio Clip Play is going to stop.
uhAtscDecAudio_ClipStPaused	Represents Audio Clip Play is paused.
uhAtscDecAudio_ClipStPausing	Represents Audio Clip Play is going to pause.

7.1.2.2.1.42 uhAtscDecAudio_ClipNtf_t

Signature

```
typedef enum _uhAtscDecAudio_ClipNtf_t {
    uhAtscDecAudio_ClipSubscriptionChanged = 0x00000001,
    uhAtscDecAudio_ClipBufferAvailable     = 0x00000002,
    uhAtscDecAudio_ClipEndOfData           = 0x00000004,
    uhAtscDecAudio_ClipStarted             = 0x00000008,
    uhAtscDecAudio_ClipStopped             = 0x00000010,
    uhAtscDecAudio_ClipPaused              = 0x00000020
} uhAtscDecAudio_ClipNtf_t, *puhAtscDecAudio_ClipNtf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAtscDecAudioClipNtf`. The values can be used to individually turn on and off the notifications.

Values

Table 130

Name	Description
uhAtscDecAudio_ClipSubscriptionChanged	Represents `OnSubscriptionChanged`.
uhAtscDecAudio_ClipBufferAvailable	Represents `OnBufferAvailable`
uhAtscDecAudio_ClipEndOfData	Represents `OnEndOfData`
uhAtscDecAudio_ClipStarted	Represents `OnStarted`
uhAtscDecAudio_ClipStopped	Represents `OnStopped`
uhAtscDecAudio_ClipPaused	Represents `OnPaused`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.1.43 uhAtscDecAudio_ClipNtfSet_t

Signature

```
typedef Int32 uhAtscDecAudio_ClipNtfSet_t, *puhAtscDecAudio_ClipNtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhAtscDec`Audio_ClipNtf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

7.1.2.2.1.44 uhAtscDecAudio_ClipAllNtfs

Signature

```
const uhAtscDecAudio_ClipNtfSet_t uhAtscDecAudio_ClipAllNtfs = 0x0000003F;
```

Qualifiers

None.

Description

Defines the set of all values of type uhAtscDec`Audio_ClipNtf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

7.1.2.2.1.45 uhAtscDecCcExtr_Ntf_t

Signature

```
typedef enum _uhAtscDecCcExtr_Ntf_t {
    uhAtscDecCcExtr_SubscriptionChanged = 0x00000001,
    uhAtscDecCcExtr_Eia708Data          = 0x00000002,
    uhAtscDecCcExtr_SelectedServiceData = 0x00000004,
    uhAtscDecCcExtr_Eia608Data          = 0x00000008
} uhAtscDecCcExtr_Ntf_t, *puhAtscDecCcExtr_Ntf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface

`uhIAtscDecCcExtrNtf`. The values can be used to individually turn on and off the notifications.

Values

Table 131

Name	Description
`uhAtscDecCcExtr_SubscriptionChanged`	Represents `OnSubscriptionChanged`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhAtscDecCcExtr_Eia708Data	Represents OnEia708Data.
uhAtscDecCcExtr_SelectedServiceData	Represents OnSelectedServiceData.
uhAtscDecCcExtr_Eia608Data	Represents OnEia608Data.

7.1.2.2.1.46 uhAtscDecCcExtr_NtfSet_t

Signature

```
typedef Int32 uhAtscDecCcExtr_NtfSet_t, *puhAtscDecCcExtr_NtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uhAtscDecCcExtr_Ntf_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

7.1.2.2.1.47 uhAtscDecCcExtr_AllNtfs

Signature

```
const uhAtscDecCcExtr_NtfSet_t uhAtscDecCcExtr_AllNtfs = 0x0000000F;
```

Qualifiers

None.

Description

Defines the set of all values of type uhAtscDecCcExtr_Ntf_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

7.1.2.2.1.48 Externally referenced types.

Please refer to 8.1.13 and 8.1.12 for understanding the

— error codes and

— types and constants

of the ATSC Demux and ATSC System Time Clock decoder.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.2.2 Model Types & Constants

7.1.2.2.2.1 AvClipBuffer

Signature

```
typedef struct {
    UInt32  bufferId;
    UInt32  bufferSize;
    pUInt8  pBuffer;
    Bool    bufferEndOfData;
} AvClipBuffer;
```

Qualifiers

None.

Description

Values of this type represent an Audio or Video Clip Buffer.

Fields

Table 132

Name	Description
`bufferId`	Identification for the buffer.
`bufferSize`	Size of the data pointed by pBuffer.
`pBuffer`	A pointer to the buffer (av clip content).
`bufferEndOfData`	Indication whether the buffer contains end of data (`True`) or not (`False`).

7.1.2.3 Logical Component

7.1.2.3.1 Interface-Role Model

Figure 23 depicts the interface role model for the ATSC Decoder logical component (the grey box in the interface-role diagram).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

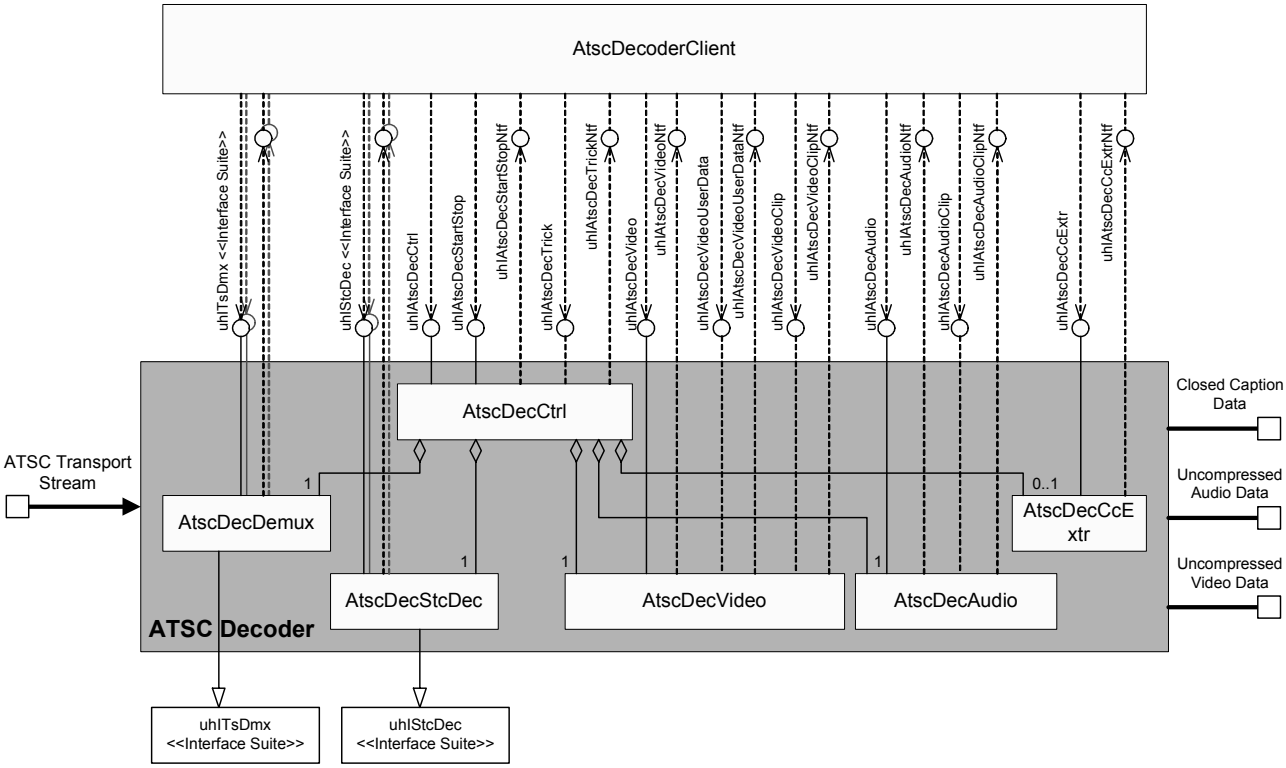


Figure 23 — Interface-Role Model of an ATSC Decoder

As shown in the figure an ATSC Decoder logical component has the following roles:

— An `AtscDecCtrl` that provides interfaces for controlling the entire ATSC Decoder logical component AtscDec. Further, it provides interfaces for accessing various roles/interfaces of the ATSC Decoder logical component. The relationship with all other roles are as depicted in Figure 23. The `AtscDecCtrl` also has interfaces for starting and stopping and for trick play.

— An `AtscDecDemux` providing interfaces for demultiplexing the incoming ATSC Transport Stream into an Audio and Video part. It also filters sections and private PES packets. It is important to note that `AtscDecDemux` role is specializing the `uhITsDmx` interface suite. Please refer to 8.1.13 for more details on the interfaces that `uhITsDmx` suite provides.

— An `AtscDecVideo` providing interfaces to decode the incoming compressed ATSC video stream. It has a mandatory `uhIAtscDecVideo` interface, which allows the user to query the incoming stream attributes and be notified of stream attributes and its changes. The optional `uhIAtscDecVideoNtf` if provided by the client is used for notification purposes. The `AtscDecVideo` has also an optional interface (`uhIAtscDecVideoUserData`) that allows to selectively extract user data present in the digital video stream. The optional interface `uhIAtscDecVideoClip` enables the client to use the video decoder for decoding video clips and provides control to the video clip play functionality. The optional `uhIAtscDecVideoClipNtf` interface enables the client to be notified of events such as the start, the end, the pausing of the video clip play.

— An `AtscDecAudio` providing interfaces to decode the incoming compressed ATSC audio stream. It has a mandatory `uhIAtscDecAudio` interface, which is used by the client to get notified of stream attributes and its changes. The optional interface `uhIAtscDecAudioNtf` if provided by the client is used for notification purposes. The optional interface `uhIAtscDecAudioClip` enables the client to use the audio decoder for decoding audio clips and provides control to the audio clip play functionality. The optional `uhIAtscDecAudioClipNtf` interface enables the client to be notified of events such as the start, the end, the pausing of the audio clip play.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— An AtscDecStcDec providing interfaces for monitoring the System Time Clock and associated alarms. It is important to note that AtscDecStcDec role is specializing the uhIStcDec interface suite. Please refer to 8.1.12 for more details on the interfaces that uhIStcDec suite provides.

— An AtscDecCcExtr providing interfaces for the client to retrieve EIA 608/708 data. It has a mandatory interface uhIAtscDecCcExtr EIA 608/708 data retrieval services. It is important to note that this role is optional.

7.1.2.3.2 Diversity

7.1.2.3.2.1 Provided Interfaces

Table 133

Role	Interface	Presence
AtscDecAudio	uhIAtscDecAudio	mandatory
AtscDecClient	uhIAtscDecAudioNtf	optional
AtscDecAudio	uhIAtscDecAudioClip	optional
AtscDecClient	uhIAtscDecAudioClipNtf	optional
AtscDecCcExtr	uhIAtscDecCcExtr	mandatory
AtscDecClient	uhIAtscDecCcExtrNtf	optional
AtscDecCtrl	uhIAtscDecCtrl	mandatory
AtscDecCtrl	uhIAtscDecStartStop	mandatory
AtscDecClient	uhIAtscDecStartStopNtf	optional
AtscDecCtrl	uhIAtscDecTrick	optional
AtscDecClient	uhIAtscDecTrickNtf	optional
AtscDecVideo	uhIAtscDecVideo	mandatory
AtscDecClient	uhIAtscDecVideoNtf	optional
AtscDecVideo	uhIAtscDecVideoUserData	optional
AtscDecClient	uhIAtscDecVideoUserDataNtf	optional
AtscDecVideo	uhIAtscDecVideoClip	optional
AtscDecClient	uhIAtscDecVideoClipNtf	optional
AtscDecStcDec	uhIStcDec <<interface suite>>	See 8.1.12
AtscDecDemux	uhITsDmx <<interface suite>>	See 8.1.13

7.1.2.3.2.2 Configurable Items

The configurable items are captured against each role of the ATSC Decoder.

Table 134

Role	Attribute
AtscDecCtrl	ccExtrSupported

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Role	Attribute
AtscDecCtrl	nrSuppSpeeds
AtscDecCtrl	suppSpeeds
AtscDecCtrl	suppFrameDecModes
AtscDecCcExtr	maxNrServices
AtscDecAudio	clipPauseSupported
AtscDecAudio	clipMaxNrBuffers
AtscDecDemux	See 8.1.13.
AtscDecStcDec	See 8.1.12.
AtscDecVideo	clipPauseSupported
AtscDecVideo	clipStepSupported
AtscDecVideo	clipMaxNrBuffers

7.1.2.3.2.3 Constraints

The constraints of this logical component are captured against each role of the ATSC Decoder.

AtscDecCtrl
- nrSuppSpeeds != 0
- Normal speed (1/1) in suppSpeeds
- uhAtscDec_AnyFrame in suppFrameDecModes

AtscDecDemux
See 8.1.13.

AtscDecStcDec
See 8.1.12.

AtscDecVideo
- if clipPauseSupported == False
  - clipStepSupported == False

AtscDecAudio
None.

AtscDecCcExtr
None.

7.1.2.3.3 Instantiation

7.1.2.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 135

Type	Object	Multiplicity
AtscDecVideo	vdec	1
AtscDecCcExtr	ccextr	0..1
AtscDecAudio	adec	1
AtscDecDemux	dmx	1
AtscDecStcDec	stcdec	1
AtscDecCtrl	ctrl	1

7.1.2.3.3.2 Initial State

The initial state of AtscDec logical component is captured in terms of its roles.

AtscDecCtrl

— if (ctrl.ccExtrSupported == False)
  — ctrl.pIAtscDecCcExtr == NULL
— ctrl.demuxState == uhAtscDecCtrl_StateStopped
— ctrl.decodingState == uhAtscDecCtrl_StateStopped
— ctrl.trickState == uhAtscDec_TrickStateStopped
— ctrl.streamMode == uhAtscDec_StreamModePush
— ctrl.speed.numerator == 1
— ctrl.speed.denominator == 1
— ctrl.pauseAtEnabled == False
— ctrl.pauseAtFirstFrame == False
— ctrl.decodingFrameMode == uhAtscDec_AnyFrame

AtscDecDemux

See 8.1.13.

AtscDecStcDec

See 8.1.12.

AtscDecVideo

— vdec.compressionStd == uhAtscDecVideo_CompStdInvalid
— vdec.level == uhAtscDecVideo_LevelInvalid
— vdec.profile == uhAtscDecVideo_InvalidProfile
— vdec.resStatus == uhAtscDec_ValStatInvalid
— vdec.height == 0
— vdec.width == 0
— vdec.frameRate == uhAtscDecVideo_InvalidFrameRate
— vdec.scanType == uhAtscDecVideo_ScanTypeInvalid
— vdec.seqDataStatus == uhAtscDec_ValStatInvalid
— vdec.gopDataStatus == uhAtscDec_ValStatInvalid
— vdec.picDataStatus == uhAtscDec_ValStatInvalid
— vdec.errorStatus == uhAtscDecVideo_NoError
— vdec.stableOutput == False
— vdec.clipCurrentState == uhAtscDecVideo_ClipStStopped

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
— vdec.avclip                == {}

AtscDecAudio
— adec.compStandard          == uhAtscDecAudio_CompStdInvalid
— adec.sampFreq              == uhAtscDecAudio_FreqInvalid
— adec.bitRateStatus         == uhAtscDec_ValStatInvalid
— adec.variableBitRate       == False
— adec.bitRate               == 0
— adec.channelStatus         == uhAtscDec_ValStatInvalid
— adec.frontChannels         == 0
— adec.rearChannels          == 0
— adec.lfeChannelPresent     == False
— adec.serviceType           == uhAtscDecAudio_ServiceInvalid
— adec.dolbySurroundPresent  == uhAtscDecAudio_DolbyInvalid
— adec.errorStatus           == uhAtscDecAudio_NoError
— adec.stableOutput          == False
— vdec.clipCurrentState      == uhAtscDecAudio_ClipStStopped
— vdec.avclip                == {}

AtscDecCcExtr
— ccextr.eia708DataStatus    == uhAtscDec_ValStatInvalid
— ccextr.selServiceDataStatus == uhAtscDec_ValStatInvalid
— ccextr.eia608DataStatus    == uhAtscDec_ValStatInvalid
— ccextr.nrServicesSelected  == 0
— For each i in {0 .. ccextr.maxNrServices - 1}
   — ccextr.servicesSelected[i] == 0
```

7.1.2.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

The `AtscDecCtrl` role will only notify `OnDecodersStopped` once all events in the `AtscDecCtrl`, `AtscDecVideo`, `AtscDecAudio`, `AtscDecStc` and the `AtscDecCcExtr` roles that occurred before the ATSC decoder entered the stopped state for decoding (`ctrl.decodingState == uhAtscDecCtrl_StateStopped`), are notified and the corresponding notification functions returned.

The `AtscDecCtrl` role will only notify `OnDemuxStopped` once all events in the `uhITsDmx` roles that occurred before the ATSC decoder entered the stopped state for the demux (`ctrl.demuxState == uhAtscDecCtrl_StateStopped`), are notified and the corresponding notification functions returned.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.4 Roles

7.1.2.4.1 AtscDecCtrl

Signature

```
role AtscDecCtrl {
    puhIAtscDecVideo                pIAtscDecVideo;
    puhIAtscDecAudio                pIAtscDecAudio;
    puhIAtscDecCcExtr               pIAtscDecCcExtr;
    puhITsDmx                       pIAtscDecDemux;
    puhIStcDec                      pIAtscDecStcDec;
    const Bool                      ccExtrSupported;
    uhAtscDecCtrl_State_t           demuxState;
    uhAtscDecCtrl_State_t           decodingState;
    uhAtscDec_TrickState_t          trickState;
    uhAtscDec_StreamMode_t          streamMode;
    const UInt32                    nrSuppSpeeds;
    const uhAtscDec_Speed_t         suppSpeeds[nrSuppSpeeds];
    uhAtscDec_Speed_t               speed;
    uhTimeStamp_t                   pauseAtPts;
    uhAtscDec_FrameType_t           pauseAtType;
    Bool                            pauseAtEnabled;
    Bool                            pauseAtFirstFrame;
    const uhAtscDec_FrameTypeSet_t  suppFrameDecModes;
    uhAtscDec_FrameType_t           frameDecMode;
}
```

Qualifiers

— root

Description

`AtscDecCtrl` provides navigational facilities for the `AtscDecClient` to access and interact with other roles of AtscDec.

Independent Attributes

Table 136

Name	Description
`pIAtscDecVideo`	Interface pointer to the ATSC Video Decoder associated with this `AtscDec`.
`pIAtscDecDemux`	Interface pointer to the ATSC Demux associated with this `AtscDec`.
`pIAtscDecAudio`	Interface pointer to the ATSC Audio Decoder associated with this `AtscDec`.
`pIAtscDecCcExtr`	Interface pointer to the ATSC Closed Caption Extractor associated with this `AtscDec`.
`pIAtscDecStcDec`	Interface pointer to the ATSC System Time Clock Decoder associated with this `AtscDec`.
`ccExtrSupported`	Indicates whether `AtscDecCcExtr` functionality is supported by `AtscDec`. If “True” `AtscDecCcExtr` functionality is supported and “False” if not supported.
`demuxState`	The current state of the `AtscDecDemux`. Figure 24 shows the corresponding state diagram. Note that the difference between push and pull, as shown in the figure, is modeled by the `streamMode` attribute.
`decodingState`	The current state of the `AtscDecAudio`, `AtscDecVideo`, `AtscDecStcDec`, and `AtscDecCcExtr`. Figure 24 shows the corresponding state diagram (replace StartDemux by StartDecoders, etc.). Note that the difference between push and pull, as shown in the figure, is modeled by the `streamMode` attribute.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
`trickState`	The current trick state of the ATSC Decoder. Figure 25 shows the corresponding state diagram. Note that the difference between push and pull, as shown in the figure, is modeled by the `streamMode` attribute.
`streamMode`	The current stream mode, being either push or pull.
`nrSuppSpeeds`	The number of supported decoding speeds.
`suppSpeeds`	Array containing the supported decoding speeds.
`speed`	The current decoding speed.
`pauseAtPts`	The pause at PTS value.
`pauseAtType`	The frame type to pause on when the PTS expires.
`pauseAtEnabled`	Indicates whether a pause at a certain PTS value has been enabled (enabled is `True`, disabled is `False`).
`pauseAtFirstFrame`	Indicates whether the decoding must pause at the first frame that is decoded when decoding is started (is `True`, if and when pause on first frame is enabled, is `False` otherwise).
`suppFrameDecModes`	The set of supported frame decoding modes.
`frameDecMode`	The current frame decoding mode.

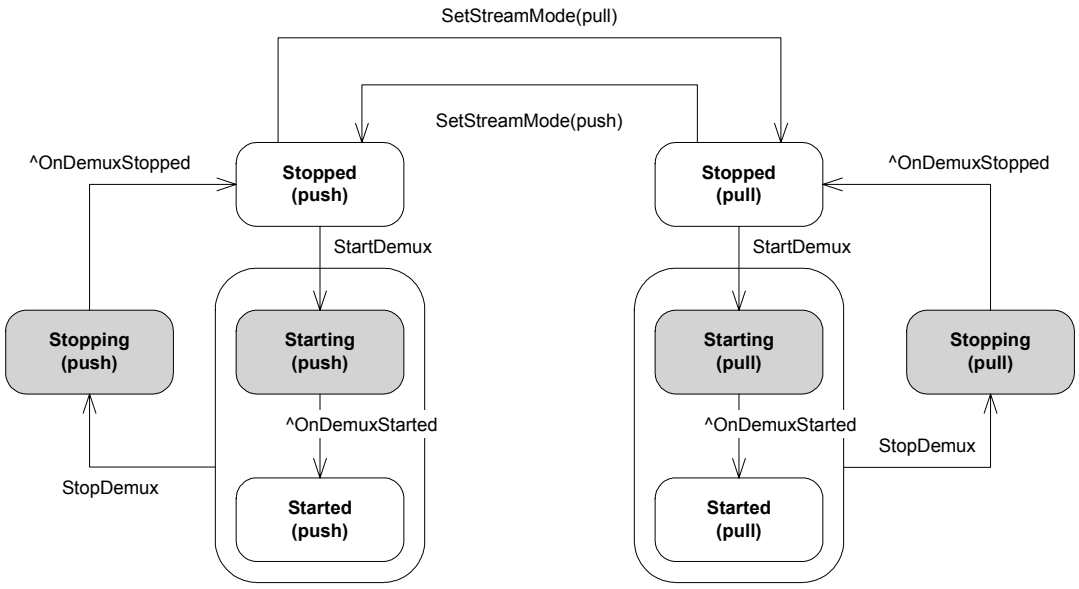


Figure 24 — The state machine for the demux (as represented by the `demuxState` attribute). The state machine for the decoding (as represented by the `decodingState` attribute is similar (replace "Demux" by "Decoders"). The colored states are transitory states.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

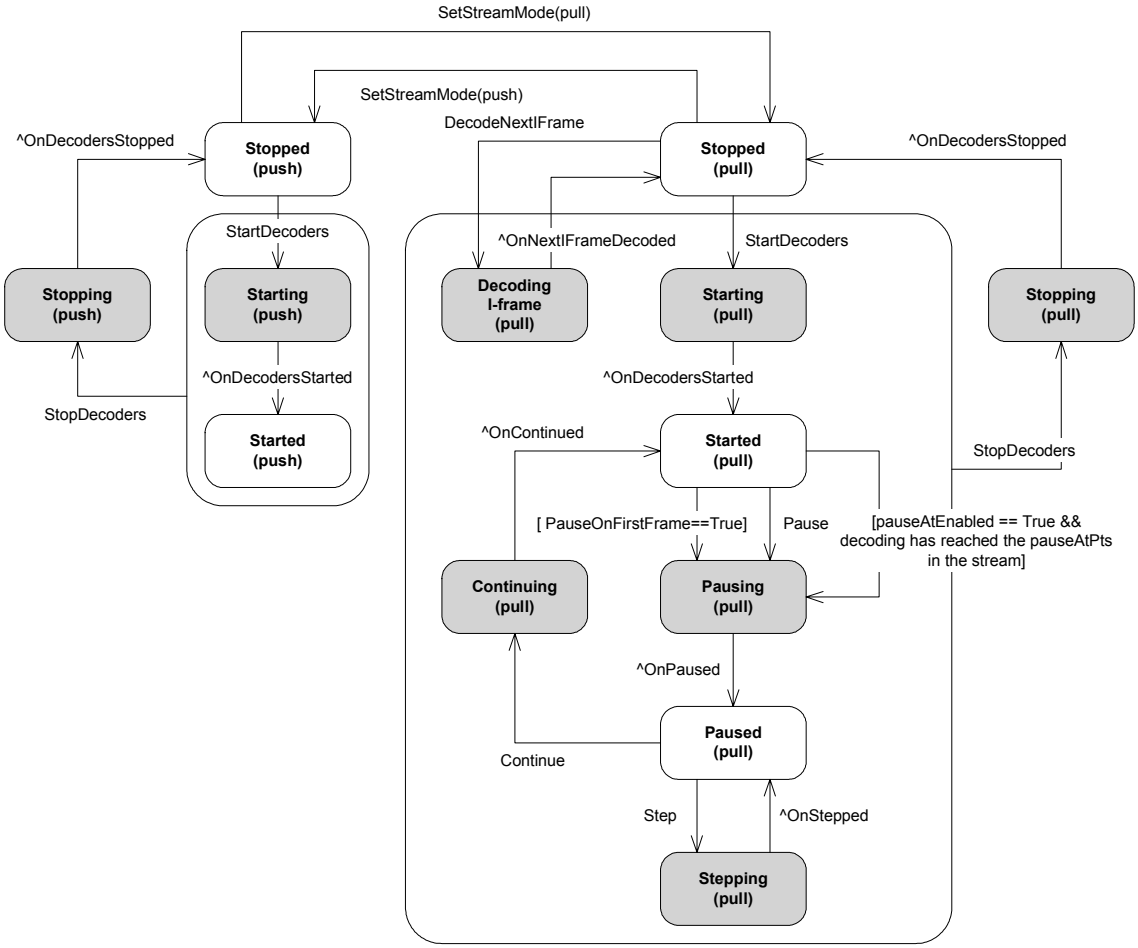


Figure 25 — The state machine for trick play decoding (as represented by the `trickState` attribute). The colored states are transitory states.

Invariants

— `frameDecMode` in `suppFrameDecModes`
— `speed` in `suppSpeeds`

Instantiation

`AtscDecCtrl` is always instantiated as part of `AtscDec`.

Streaming Behavior

None.

Active Behavior

The `AtscDecCtrl` has standard notification behavior (see `uhIAtscDecStartStopNtf` and `uhIAtscDecTrickNtf`), apart from the following events.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 137

Event	Action
Only if `ctrl.pauseAtEnabled == True`: The decoding reaches a PTS in the stream that matches `ctrl.pauseAtPts`.	`ctrl.trickState = uhAtscDec_TrickStatePausing`

Only if `ctrl.pauseAtEnabled == True` and `ctrl.trickState == uhAtscDec_TrickStatePausing`:
The first frame with a frame type that matches `ctrl.pauseAtType` has been decoded.
	`ctrl.pauseAtEnabled = False`
	`ctrl.trickState = uhAtscDec_TrickStatePaused`
	— notify `OnPaused(*)`

`Only if ctrl.pauseAtFirstFrame == True:` `ctrl.trickState` becomes equal to uhAtscDec`_TrickStateStarted`.	`ctrl.trickState = uhAtscDec_TrickStatePausing`
Only if `ctrl.pauseAtFirstFrame == True` and `ctrl.trickState == uhAtscDec_TrickStatePausing`: The first frame has been decoded.	ctrl.pauseAtFirstFrame = False `ctrl.trickState = uhAtscDec_TrickStatePaused` — notify `OnPaused(*)`

7.1.2.4.2 AtscDecDemux

Signature

```
role AtscDecDemux specialises uhITsDmx {}
```

Qualifiers

— See 8.1.13 for further details.

Description

The `AtscDecDemux` specialises the context of `uhITsDmx` in the sense that it is being used in the ATSC Decoder context. Since it is being used in the ATSC Decoder context the root interface of `uhITsDmx` interface suite as specified in 8.1.13 is not applicable here. For further details on `uhITsDmx` interface suite please refer to 8.1.13.

Independent Attributes

See 8.1.13 for further details.

Invariants

— See 8.1.13 for further details.

Instantiation

`AtscDecDemux` is always instantiated as part of `AtscDec`.

Streaming Behavior

See 8.1.13 for further details. In addition, if the `AtscDecDemux` is in the Stopped state (`ctrl.demuxState == uhAtscDecCtrl_StateStopped`), then it does not process its input and it indicates that its sync lock is invalid. In the Stopped state, the `AtscDecDemux` retains its state (e.g. section filters allocated by the client, or selected PID values).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Active Behavior
See 8.1.13 for further details.

7.1.2.4.3 AtscDecStcDec

Signature
```
role AtscDecStcDec specialises uhIStcDec {}
```

Qualifiers
— See 8.1.12 for further details.

Description
`AtscDecStcDec` specialises the context of `uhIStcDec` in the sense that it is being used in the ATSC Decoder context. Since it is being used in the ATSC Decoder context the root interface of `uhIStcDec` interface suite as specified in 8.1.12 is not applicable here. For further details on `uhIStcDec` interface suite please refer to 8.1.12.

Independent Attributes
See 8.1.12 for further details.

Invariants
— See 8.1.12 for further details.

Instantiation
`AtscDecStcDec` is always instantiated as part of AtscDec.

Streaming Behavior
See 8.1.12 for further details.

Active Behavior
See 8.1.12 for further details.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.4.4 AtscDecVideo

Signature

```
role AtscDecVideo {
    uhAtscDecVideo_CompStd_t        compressionStd;
    uhAtscDecVideo_Level_t          level;
    uhAtscDecVideo_Profile_t        profile;
    uhAtscDec_ValStat_t             resStatus;
    UInt32                          height;
    UInt32                          width;
    uhAtscDecVideo_FrameRate_t      frameRate;
    uhAtscDecVideo_ScanType_t       scanType;
    uhAtscDec_ValStat_t             seqDataStatus;
    uhAtscDec_ValStat_t             gopDataStatus;
    uhAtscDec_ValStat_t             picDataStatus;
    uhAtscDecVideo_ErrorSet_t       errorStatus;
    Bool                            stableOutput;
    uhAtscDecVideo_ClipState_t      clipCurrentState;
    const Bool                      clipPauseSupported;
    const Bool                      clipStepSupported;
    const UInt32                    clipMaxNrBuffers;
    setof{AvClipBuffer}             avclip;
    UInt32                          clipNrOfCurrentBuffers;
    Bool                            consumingData;
}
```

Qualifiers

None.

Description

`AtscDecVideo` decodes an ATSC encoded digital video input stream, or a video clip provided by the client, and provides the decoded video stream at its output. `AtscDecVideo` detects the compression standards that match the decoder's capability and provides notifications of various changes that occur in the input stream (or clip input). It also extracts the picture user data present in the encoded digital video input stream (or clip input).

Independent Attributes

Table 138

Name	Description
`compressionStd`	Compression standard of the current video stream
`level`	MPEG-2 Level of the current video stream
`profile`	MPEG-2 Profile of the current video stream
`resStatus`	The validity status of the resolution (height and width) of the pictures in the current video stream.
`height`	The height of the picture in the current video stream in units of pixels.
`width`	The width of the picture in the current video stream in units of pixels.
`frameRate`	The frame rate of pictures in the input stream.
`scanType`	The way pictures are scanned in the input stream.
`seqDataStatus`	The validity status of the sequence user data and related properties.
`gopDataStatus`	The validity status of the GOP user data and related properties.
`picDataStatus`	The validity status of the picture user data and related properties.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
errorStatus	The current error status, represented by the set of errors that have occurred, and from which the video decoder has not yet recovered.
stableOutput	Indicates whether the video decoder is outputting decoded video long enough so that the client can safely unblank the corresponding output at the appropriate logical component(s) downstream (True if this is the case, False otherwise). NOTE 1 While in a trick mode (like paused, or stepping), the video decoder is considered to be outputting decoded video.
clipCurrentState	Indicates the current status of ATSC Video Clip play.
clipPauseSupported	Flag indicating whether Pause functionality is supported by ATSC Video decoder in the video clip play mode.
clipStepSupported	Flag indicating whether Step functionality is supported by ATSC Video decoder in the video clip play mode.
clipMaxNrBuffers	Indicates maximum number of video clip data buffers that the video decoder can hold at the same time.
avclip	A set of video clip buffers. The clip buffers are decoded in the same order as they were added by the client.

Dependent attributes

Table 139

Name	Description
clipNrOfCurrentBuffers	Indicates the number of video clip data buffers in avclip.
consumingData	Indication, in the clip play mode, whether AtscDecVideo is consuming data from buffers available in avclip. It is True when AtscDecVideo is consuming data from buffers available in avclip. It is False, otherwise. Consumption of the data by the decoder depends on its state: in the uhAtscDecVideo_ClipStStarted state, it is consuming data; in the uhAtscDecVideo_ClipStPaused and uhAtscDecVideo_ClipStStopped states, it is not consuming data; in the uhAtscDecVideo_ClipStStarting, uhAtscDecVideo_ClipStPausing, and uhAtscDecVideo_ClipStStopping states, the value of the consumingData attribute stays identical to the value it had in the previous state.

Invariants

- (ctrl.decodingState == uhAtscDecCtrl_StateStopped) implies (vdec.stableOutput == False)
- clipNrOfCurrentBuffers <= clipMaxNrBuffers
- if (!clipPauseSupported)
  - clipCurrentState != uhAtscDecVideo_ClipStPaused
  - clipCurrentState != uhAtscDecVideo_ClipStPausing

Instantiation

AtscDecVideo is always instantiated as part of an ATSC Decoder.

Streaming Behavior

- When trying to decode its input, the AtscDecVideo continuously observes its video input stream. When a video input stream is detected AtscDecVideo will try to detect the compression standard of the input stream and set the value of compressionStd accordingly.
  As soon as the compression standard of the input stream is MPEG-2 then compressionStd is set to uhAtscDecVideo_CompStdMpeg2 and the input stream is decoded. At all other times the value of compressionStd is equal to uhAtscDecVideo_CompStdInvalid (when there is no input stream) or

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

uhAtscDecVideo_CompStdUnknown (when there is an input stream, but the compression standard could not be determined) but AtscDecVideo continues to observe its video input stream for a valid compression standard.

— When one or more changes occur in the properties of the video input stream, AtscDecVideo provides a notification to the subscribed client(s).

— When an error occurs during the decoding, the subscribed client(s) are notified. The AtscDecVideo will try to recover, and when recovered it will notify the subscribed client(s) again.

— When clip play decoding is started, decoding data from the transport stream demultiplexer will be suspended until clip play decoding is stopped. Clip play decoding has preference.

— When the video decoder is decoding clip data, it will decode the clip data in the order it has been added.

— If the video decoder does not output a newly decoded frame for whatever reason (e.g., the decoding is stopped, the decoder has encountered an error, or the decoding is paused), the default behaviour is that the platform will repeat the last decoded frame or field for the downstream components.

Active Behavior

The AtscDecVideo has standard notification behavior. (see uhIAtscDecVideoNtf), apart from the following events.

Table 140

Event	Action
A buffer with clip data has been decoded.	— Let 'b' be the buffer that has just been decoded — Remove 'b' from vdec.avclip — Notify OnBufferAvailable(*) — If (b.bufferEndOfData == True) — Notify OnEndOfData(*)
A clip buffer with bufferSize==0 and bufferEndOfData ==True has been encountered while decoding clip buffers (i.e. the clip buffer would have been decoded if bufferSize != 0).	— The clip buffer is removed from vdec.avclip — Notify OnEndOfData(*)
Only if (clipCurrentState == uhAtscDecVideo_ClipStStopped) || ((clipCurrentState != uhAtscDecVideo_ClipStStopped) && consumingData):	— Let *e* be the set of the one or more errors that the video decoder encountered and that are not part of vdec.errorStatus. — Add the errors in *e* to vdec.errorStatus
The video decoder encounters one or more errors that are not part of vdec.errorStatus. NOTE If clipCurrentState != uhAtscDecVideo_ClipStStopped and the bufferEndOfData flag of the last decoded or encountered buffer is True, then the video decoder will *not* treat a possible underrun as an error situation.	— Notify OnError(*, e)

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Event	Action
The video decoder recovers from one or more errors. NOTE 1 It is up to the implementation to decide when it has recovered from an error. NOTE 2 When video clip play starts or stops, the video decoder "recovers" from all errors if there were any (and then the OnRecovered event is notified).	— Let *e* be the set of the one or more errors that the video decoder recovered from. — Remove the errors in *e* from `vdec.errorStatus` — Notify `OnRecovered(*, e)`
The video decoder is outputting decoded video long enough so that the client can safely unblank the corresponding output at the appropriate logical component(s) downstream. (`vdec.stableOutput == False` when this event occurs).	`vdec.stableOutput = True` — Notify `OnStableOutput(*, True)`
The video decoder is no longer outputting a stable (decoded) video stream (`vdec.stableOutput == True` when this event occurs).	`vdec.stableOutput = False` — Notify `OnStableOutput(*, False)`

7.1.2.4.5 AtscDecAudio

Signature

```
role AtscDecAudio {
    uhAtscDecAudio_CompStd_t           compStandard;
    uhAtscDecAudio_SampFreq_t          sampFreq;
    uhAtscDec_ValStat_t                bitRateStatus;
    Bool                               variableBitRate;
    UInt32                             bitRate;
    uhAtscDec_ValStat_t                channelStatus
    UInt32                             frontChannels;
    UInt32                             rearChannels;
    Bool                               lfeChannelPresent;
    uhAtscDecAudio_ServiceType_t       serviceType;
    uhAtscDecAudio_DolbySurroundInfo_t dolbySurroundPresent;
    uhAtscDecAudio_ErrorSet_t          errorStatus;
    Bool                               stableOutput;
    uhAtscDecAudio_ClipState_t         clipCurrentState;
    const Bool                         clipPauseSupported;
    const UInt32                       clipMaxNrBuffers;
    setof{AvClipBuffer}                avclip;
    UInt32                             clipNrOfCurrentBuffers;
    Bool                               consumingData;
}
```

Qualifiers

None.

Description

`AtscDecAudio` decodes an ATSC encoded audio input stream, or an audio clip provided by the client, and outputs a decoded audio stream. It autonomously detects the compression standard of the encoded audio input stream (or clip input) and provides notifications of various changes that occur in the input stream (or clip input).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 141

Name	Description
compStandard	The audio compression standard of the digital audio input stream.
sampFreq	The sampling frequency of the digital audio input stream.
bitRateStatus	The validity status of the bit rate related properties (variableBitRate, bitRate).
variableBitRate	Indicates whether the digital audio input stream has a variable bit rate.
bitRate	The bit rate (in kbit/s) of the digital audio input stream. When the digital audio input stream has a variable bit rate or when the bit rate of the digital audio input stream is unknown, bitRate == 0.
channelStatus	The validity status of the audio channel configuration.
frontChannels	The number of front channels in the digital audio input stream.
rearChannels	The number of rear channels in the digital audio input stream.
lfeChannelPresent	Indicates whether a LFE channel is present in the digital audio input stream.
serviceType	The type of audio service that the digital audio input stream conveys. When the type of the audio service is not present in the input stream, serviceType == uhAtscDecAudio_ServiceUnknown.
dolbySurroundPresent	Information about the presence of Dolby Surround in the digital audio input stream.
errorStatus	The current error status, represented by the set of errors that have occurred, and from which the audio decoder has not yet recovered.
stableOutput	Indicates whether the audio decoder is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream (True if this is the case, False otherwise). NOTE 1 While in a trick mode (like paused, or decoding with a different speed than normal speed), the audio decoder is considered to be outputting decoded audio (although this may be “silence”).
clipCurrentState	Indicates the current status of audio clip play.
clipPauseSupported	Flag indicating whether Pause functionality is supported by ATSC Audio decoder in the audio clip play mode.
clipMaxNrBuffers	Indicates maximum number of audio clip data buffers that client(s) can add at a the same time.
avclip	A set of audio clip buffers. The clip buffers are decoded in the same order as they were added by the client.

Dependent Attributes

Table 142

Name	Description
clipNrOfCurrentBuffers	Indicates the number of audio clip data buffers in avclip.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
consumingData	Indication, in the clip play mode, whether AtscDecAudio is consuming data from buffers available in avclip. It is True when AtscDecAudio is consuming data from buffers available in avclip. It is False, otherwise. Consumption of the data by the decoder depends on its state: in the uhAtscDecAudio_ClipStStarted state, it is consuming data; in the uhAtscDecAudio_ClipStPaused and uhAtscDecAudio_ClipStStopped states, it is not consuming data; in the uhAtscDecAudio_ClipStStarting, uhAtscDecAudio_ClipStPausing, and uhAtscDecAudio_ClipStStopping states, the value of the consumingData attribute stays identical to the value it had in the previous state.

Invariants

- If variableBitRate == True
  - bitRate == 0
- If compStandard == uhAtscDecAudio_CompStdUnknown
  - sampFreq == uhAtscDecAudio_FreqUnknown
- If sampFreq == uhAtscDecAudio_FreqUnknown
  - variableBitRate = False
  - bitRate == 0
  - frontChannels == 0
  - rearChannels == 0
  - lfeChannelPresent == False
  - serviceType == uhAtscDecAudio_ServiceUnknown
  - dolbySurroundPresent == uhAtscDecAudio_DolbyUnknown
  - (ctrl.decodingState == uhAtscDecCtrl_StateStopped) implies (adec.stableOutput == False)
  - clipNrOfCurrentBuffers <= clipMaxNrBuffers
- if (!clipPauseSupported)
  - clipCurrentState != uhAtscDecVideo_ClipStPaused
  - clipCurrentState != uhAtscDecVideo_ClipStPausing

Instantiation

AtscDecAudio is always instantiated as part of an ATSC Decoder.

Streaming Behavior

The following streaming behavior applies when the streaming input of the AtscDecAudio comes from the demultiplexer:

- When trying to decode its input, the AtscDecAudio continually observes the input stream. When an input stream is detected AtscDecAudio will try to detect the compression standard of the input stream and set the value compStandard and sampFreq accordingly. As long as no compression standard has been detected the value of compStandard is equal to uhAtscDecAudio_CompStdUnknown (in case there is an input stream) or uhAtscDecAudio_CompStdInvalid (in case there is no input stream). AtscDecAudio will only detect standards that are in the set of compression standards supported in the ATSC standard, i.e. AC-3 only. As soon as the AC-3 standard has been detected, the value of compStandard is set to uhAtscDecAudio_CompStdAc3.

- If the sampling frequency *sf* of the audio input stream is not in the set of sampling frequencies supported (i.e. 48 kHz in the case of the ATSC standard), the value of sampFreq remains set to uhAtscDecAudio_FreqUnknown. As soon as the sampling frequency *sf* of the input stream is 48 kHz, the value of sampFreq is set to uhAtscDecAudio_48k and the input stream is decoded. In case there is no input stream, the sampling frequency *sf* is set to uhAtscDecAudio_FreqInvalid.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— When the compression standard is not AC-3 (anymore), the value of `compStandard` is (re)set to uh`AtscDecAudio_CompStdUnknown` and `sampFreq` is (re)set to uh`AtscDecAudio_FreqUnknown`. When there is no input stream, the value of `compStandard` is (re)set to uh`AtscDecAudio_CompStdInvalid` and `sampFreq` is (re)set to uh`AtscDecAudio_FreqInvalid`. The `AtscDecAudio` keeps reading and parsing the incoming audio packets, looks respectively for a valid compression standard and for a valid sampling frequency, and no output data is produced until a valid compression standard and a valid sampling frequency have been detected.

— When one or more changes occur in the properties of the audio input stream, `AtscDecAudio` provides a notification to the subscribed client(s). If the changes in the properties leads to artifacts for e.g. blops, ticks then `AtscDecAudio` stops producing output data and a soft mute (a cosine mute, for instance) is applied. `AtscDecAudio` determines when it is possible to resume again and a soft unmute (a cosine unmute, for instance) is accordingly applied.

— When an error occurs during the decoding, the subscribed client(s) are notified. The `AtscDecAudio` will try to recover, and when recovered it will notify the subscribed client(s) again.

When decoding an audio clip, the above streaming behavior also applies, but with the following difference: MPEG-1 layer 1, 2 and 3 are also supported, and the sampling frequencies 32kHz, 44.1kHz and 48kHz are also supported for MPEG-1 layer 1, 2 and 3.

— When audio clip play decoding is started, decoding data from the transport stream demultiplexer will be suspended until clip play decoding is stopped. Clip play decoding has preference.

— When the audio decoder is decoding clip data, it will decode the clip data in the order it has been added.

Active Behavior

`AtscDecAudio` has standard notification behaviour (see `uhIAtscDecAudioNtf`), apart from the following events.

Table 143

Event	Action
A buffer with clip data has been decoded.	— Let '`b`' be the buffer that has just been decoded — Remove '`b`' from `adec.avclip` — Notify `OnBufferAvailable(*)` — If (`b.bufferEndOfData == True`) — Notify `OnEndOfData(*)`
A clip buffer with `bufferSize==0` and `bufferEndOfData ==True` has been encountered while decoding clip buffers (i.e. the clip buffer would have been decoded if `bufferSize != 0`).	— The clip buffer is removed from `adec.avclip` — Notify `OnEndOfData(*)`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Event	Action
Only if `(clipCurrentState == uhAtscDecAudio_ClipStStopped) || ((clipCurrentState != uhAtscDecAudio_ClipStStopped) && consumingData)`:	— Let *e* be the set of the one or more errors that the audio decoder encountered and that are not part of `adec.errorStatus`. — Add the errors in *e* to `adec.errorStatus` — Notify `OnError(*, e)`
The audio decoder encounters one or more errors that are not part of `adec.errorStatus`. NOTE If `clipCurrentState != uhAtscDecAudio_ClipStStopped` and the `bufferEndOfData` flag of the last decoded or encountered buffer is True, then the audio decoder will *not* treat a possible underrun as an error situation.	
The audio decoder recovers from one or more errors. NOTE 1 It is up to the implementation to decide when it has recovered from an error. NOTE 2 When audio clip play starts or stops, the audio decoder "recovers" from all errors if there were any (and then the OnRecovered event is notified).	— Let *e* be the set of the one or more errors that the audio decoder recovered from. — Remove the errors in *e* from `adec.errorStatus` — Notify `OnRecovered(*, e)`
The audio decoder is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream. (`adec.stableOutput == False` when this event occurs).	`adec.stableOutput = True` — Notify `OnStableOutput(*, True)`
The audio decoder is no longer outputting a stable (decoded) audio stream (`adec.stableOutput == True` when this event occurs).	`adec.stableOutput = False` — Notify `OnStableOutput(*, False)`

7.1.2.4.6 AtscDecCcExtr

Signature

```
role AtscDecCcExtr {
    const UInt32        maxNrServices;
    UInt32              nrServicesSelected;
    UInt32              servicesSelected[maxNrServices];
    uhAtscDec_ValStat_t eia708DataStatus;
    uhAtscDec_ValStat_t selServiceDataStatus;
    uhAtscDec_ValStat_t eia608DataStatus;
}
```

Qualifiers

None.

Description

`AtscDecCcExtr` monitors the user data input stream and extracts Closed Caption data from it. It provides notifications to the subscribed client(s):

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— when the end of an EIA-708 digital CC packet has been detected. The DTVCC data is notified in the appropriate order for further decoding (see subclause 4.4.1 of [15]) to the subscribed client(s). See Figure 26 for the organization of the notified data.

— when EIA-708 digital Closed Caption data from a selected Caption Service has been extracted. CC Service Blocks, as described in clause 6 of [15], are notified in the appropriate order for further decoding (see subclause 4.4.1 of [15]) to the subscribed client(s).

— when analog CC data (NTSC line 21, odd and/or even field) has been extracted from the user data stream. The bit layout of the notified analog CC data follows the CC standard (see [14]); this is shown in Figure 27.

`AtscDecCcExtr` starts passing on closed caption data to a client only after this client has subscribed to the appropriate notification.

Independent Attributes

Table 144

Name	Description
`maxNrServices`	The maximum number of close caption services for which CC Service Blocks can be notified via the function `OnSelectedServiceData` to `AtscDecClient`.
`nrServicesSelected`	The number of close caption service for which the corresponding CC Service Block(s) are notified via the `OnSelectedServiceData` function to `AtscDecClient`, if subscribed.
`servicesSelected`	The set of close caption services for which CC Service Blocks are notified via the `OnSelectedServiceData` function to `AtscDecClient`, if subscribed. The services are identified by their service number
`eia708DataStatus`	The validity status of the EIA-708 CC data and related properties.
`selServiceDataStatus`	The validity status of the CC service block data and related properties.
`eia608DataStatus`	The validity status of the EIA-608 CC data and related properties.

Dependent Attributes
None.

Invariants
— `nrServicesSelected ≤ maxNrServices`

Instantiation
`AtscDecCcExtr` role is instantiated if Closed Caption functionality is supported by `AtscDec`. Please refer to subclause 7.1.2.3.3.2 for more information.

Streaming Behavior
When trying to extract data from its input, the `AtscDecCcExtr` continually monitors its user data input stream and extracts the data payload related to closed caption data. It passes on a data stream containing either the analog EIA-608 CC data (NTSC line 21, odd and/or even field) extracted from the user data input stream or the digital EIA-708 CC data extracted from the user data input stream. The output data stream follows EIA-608 standard (see [14]) or EIA-708 standard (see [15]) respectively.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Active Behavior
`AtscDecCcExtr` has standard notification behavior (see

`uhIAtscDecCcExtrNtf`).

Remarks
`AtscDecCcExtr` passes CC service blocks with service numbers as defined in `servicesSelected` to `AtscDecClient` via `OnSelectedServiceData` notification.

7.1.2.4.7 AtscDecClient

Signature
`role` `AtscDecClient {}`

Qualifiers
— actor

Description
`AtscDecClient` represents a user of all interfaces provided as part of ATSC Decoder. It can provide the optional notification interfaces for receiving events generated by the roles of (`AtscDecDemux`, `AtscDecStcDec`, `AtscDecVideo`, `AtscDecAudio`, `AtscDecCcExtr`) of ATSC Decoder. No constraints apply to the role of `AtscDecClient` other than those implied by Figure 23.

7.1.2.5 Interfaces

7.1.2.5.1 AtscDec::uhITsDmx

Qualifiers
None.

Description
This interface of AtscDec is a specialization of `uhITsDmx` interface suite 8.1.13. It only specialises the context under which `uhITsDmx` is being used i.e. ATSC Decoder. It is also important to note that since `uhITsDmx` is being used under the context of ATSC decoder the root role of `uhITsDmx` as defined in 8.1.13 is not applicable here.

Interface ID
`See 8.1.13.`

7.1.2.5.2 AtscDec::uhIStcDec

Qualifiers
None.

Description
This interface of `AtscDec` is a specialization of `uhIStcDec` interface suite 8.1.12. It only specialises the context under which `uhIStcDec` is being used i.e. ATSC Decoder. It is also important to note that since

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

uhIStcDec is being used under the context of ATSC decoder the root role of uhIStcDec as defined in 8.1.12 not applicable here.

Interface ID
See 8.1.12.

7.1.2.5.3 uhIAtscDecCtrl

Qualifiers
— root

Description
This interface is provided by AtscDec for controlling and navigating across various roles of ATSC Decoder.

Interface ID
uuid(7E2F8D2C-1CFA-473b-8CC3-7C135CB7AA9E)

7.1.2.5.3.1 GetAtscDecDemux

Signature
```
uhErrorCode_t GetAtscDecDemux ([out] uhITsDmx **ppIAtscTsDmx );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns an interface pointer to uhITsDmx. Clients can use this interface pointer to further exercise control on AtscDecDemux.

Parameters

Table 145

Name	Description
ppIAtscTsDemux	Output parameter returning an interface pointer to uhITsDmx. See 8.1.13.

Return Values
Standard.

Precondition
True

Action
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `ppIAtscTsDemux == ctrl.pIAtscDecDemux`

7.1.2.5.3.2 GetAtscDecStcDec

Signature
```
uhErrorCode_t GetAtscDecStcDec ([out] uhIStcDec **ppIAtscDecStcDec );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns an interface pointer to `uhIStcDec`. Clients can use this interface pointer to further exercise control on `AtscDecStcDec`.

Parameters

Table 146

Name	Description
`ppIAtscDecStcDec`	Output parameter returning an interface pointer to `uhIStcDec`. See 8.1.12.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `ppIAtscDecStcDec == ctrl.pIAtscDecStcDec`

7.1.2.5.3.3 GetAtscDecVideo

Signature
```
uhErrorCode_t GetAtscDecVideo ([out] uhIAtscDecVideo **ppIAtscDecVideo );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns an interface pointer to `uhIAtscDecVideo`. Clients can use this interface pointer to further exercise control on `AtscDecVideo`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 147

Name	Description
ppIAtscDecVideo	Output parameter returning an interface pointer to uhIAtscDecVideo.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— ppIAtscDecVideo == ctrl.pIAtscDecVideo

7.1.2.5.3.4 GetAtscDecAudio

Signature
```
uhErrorCode_t GetAtscDecAudio ([out] uhIAtscDecAudio **ppIAtscDecAudio );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns an interface pointer to uhIAtscDecAudio. Clients can use this interface pointer to further exercise control on AtscDecAudio.

Parameters

Table 148

Name	Description
ppIAtscDecAudio	Output parameter returning an interface pointer to uhIAtscDecAudio.

Return Values
Standard.

Precondition
True

Action
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— ppIAtscDecAudio == ctrl.pIAtscDecAudio

7.1.2.5.3.5 GetAtscDecCcExtr

Signature
uhErrorCode_t GetAtscDecCcExtr ([out] uhIAtscDecCcExtr **ppIAtscDecCcExtr);

Qualifiers
— synchronous

— thread-safe

Description
Returns an interface pointer to uhIAtscDecCcExtr if AtscDecCcExtr functionality is supported by this AtscDec. Clients can use this interface pointer to further exercise control on AtscDecCcExtr.

Parameters

Table 149

Name	Description
ppIAtscDecCcExtr	Output parameter returning an interface pointer to uhIAtscDecCcExtr.

Return Values
Only non-standard error values are listed here.

Table 150

Value	Description
UH_ERR_ATSCDEC_NO_CCEXTR	This AtscDec does not support AtscDecCcExtr functionality.

Precondition
True

Action
Modify ppIAtscDecCcExtr

Postcondition
— ppIAtscDecCcExtr == ctrl.pIAtscDecCcExtr

Remarks
If ctrl.ccExtrSupported == False then the interface pointer will be NULL.

7.1.2.5.4 uhIAtscDecStartStop

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface of `AtscDecCtrl` provides functions to start and stop the de-multiplexer and the decoders.

Interface ID
`uuid( B9E2D7B2-7365-4370-9E13-1DB8FAEB84B6 )`

7.1.2.5.4.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAtscDecStartStopNtf          *pINotify,
    [in] UInt32                           cookie,
    [in] uhAtscDecCtrl_StartStopNtfSet_t  notifs );
```

Qualifiers
— subscribe-function

7.1.2.5.4.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAtscDecStartStopNtf          *pINotify,
    [in] UInt32                           cookie,
    [in] uhAtscDecCtrl_StartStopNtfSet_t  notifs );
```

Qualifiers
— unsubscribe-function

7.1.2.5.4.3 StartDemux

Signature
```
uhErrorCode_t StartDemux ();
```

Qualifiers
— asynchronous

— single-threaded

Description
Starts the de-multiplexer.

NOTE If `ctrl.demuxState` changes to uhAtscDec`Ctrl_StateStopping` or uhAtscDec`Ctrl_StateStopped` before the execution of the asynchronous part of this function has started, the asynchronous part will not be executed. Always either all or nothing of the asynchronous part is executed.

Parameters
None.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`ctrl.demuxState == uhAtscDecCtrl_StateStopped`

`ctrl.streamMode` is compatible with the input stream and the settings in the system (see subclause 7.1.2.1.6)

Action
— Modify `ctrl.demuxState`

Postcondition
`ctrl.demuxState == uhAtscDecCtrl_StateStarting`

Asynchronous Action
Modify `ctrl.demuxState`

`Notify OnDemuxStarted`(*)

Asynchronous Postcondition
`ctrl.demuxState == uhAtscDecCtrl_StateStarted`

7.1.2.5.4.4 StopDemux

Signature
```
uhErrorCode_t StopDemux ();
```

Qualifiers
— asynchronous

— single-threaded

Description
Stops the de-multiplexer and flushes the de-multiplexer input buffer, the audio input buffer and the video input buffer. All state in the de-multiplexer that does not directly depend on the input signal, is kept (e.g. all section filters allocated by the client will become active again when the de-multiplexer is started again and a started PID filter remains started, although it will not output any data while the de-multiplexer is in the stopped state).

Parameters
None.

Return Values
Standard.

Precondition
— `!(ctrl.demuxState == uhAtscDecCtrl_StateStopped)`
— `!(ctrl.demuxState == uhAtscDecCtrl_StateStopping)`

Action
— Modify `ctrl.demuxState`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `ctrl.demuxState == uhAtscDecCtrl_StateStopping`

Asynchronous Action
— Modify `ctrl.demuxState`
— Notify `OnDemuxStopped(*)`

Asynchronous Postcondition
— `ctrl.demuxState == uhAtscDecCtrl_StateStopped`
— The input buffer of the de-multiplexer is empty (has been flushed).
— The audio input buffer and video input buffer are empty (have been flushed).

Remarks
While the de-multiplexer is in the Stopped state, it will indicate that its sync lock is invalid.

7.1.2.5.4.5 GetDemuxState

Signature
```
uhErrorCode_t GetDemuxState (
    [out] puhAtscDecCtrl_State_t pDemuxState );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the operational state of the de-multiplexer.

Parameters

Table 151

Name	Description
`pDemuxState`	Output parameter returning the current de-multiplexer state.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pDemuxState == ctrl.demuxState`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.4.6 StartDecoders

Signature
`uhErrorCode_t StartDecoders ();`

Qualifiers
— asynchronous

— single-threaded

Description
Starts the decoders (audio decoder, video decoder, STC decoder and CC extractor).

NOTE If `ctrl.decodingState` changes to uhAtscDec`Ctrl_StateStopping` or uhAtscDec`Ctrl_StateStopped` before the execution of the asynchronous part of this function has started, the asynchronous part will not be executed. Always either all or nothing of the asynchronous part is executed.

Parameters
None.

Return Values
Standard.

Precondition
— `ctrl.decodingState == uhAtscDecCtrl_StateStopped`
— `ctrl.streamMode` is compatible with the input stream and the settings in the system (see subclause 7.1.2.1.6)

Action
— Modify `ctrl.decodingState`
— Modify `ctrl.trickState`

Postcondition
— `ctrl.decodingState == uhAtscDecCtrl_StateStarting`
— `ctrl.trickState == uhAtscDec_TrickStateStarting`

Asynchronous Action
— Modify `ctrl.decodingState`
— Modify `ctrl.trickState`
— Notify `OnDecodersStarted(*)`

Asynchronous Postcondition
— `ctrl.decodingState == uhAtscDecCtrl_StateStarted`
— `ctrl.trickState == uhAtscDec_TrickStateStarted`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.4.7 StopDecoders

Signature
```
uhErrorCode_t StopDecoders ();
```

Qualifiers
- asynchronous
- single-threaded

Description
Stops the decoders (audio decoder, video decoder, STC decoder and CC extractor). The error status is reset. Any other state in the decoders that does not directly depend on the input signal, is kept.

Parameters
None.

Return Values

Table 152

Value	Description
`UH_ERR_ATSCDEC_STOP_IGNORED`	The request to stop has been ignored. This error occurs if and only if: `(ctrl.decodingState == uhAtscDecCtrl_StateStopped) || (ctrl.decodingState == uhAtscDecCtrl_StateStopping)`

Precondition
- `vdec.clipCurrentState == uhAtscDecVideo_ClipStStopped`
- `adec.clipCurrentState == uhAtscDecAudio_ClipStStopped`

Action
- Modify `ctrl.decodingState`
- Modify `ctrl.trickState`

Postcondition
- `ctrl.decodingState == uhAtscDecCtrl_StateStopping`
- `ctrl.trickState == uhAtscDec_TrickStateStopping`

Asynchronous Action
- Modify `ctrl.decodingState`
- Modify `ctrl.trickState`
- Modify `ctrl.pauseAtEnable`
- Modify `ctrl.pauseAtFirstFrame`
- Modify `adec.errorStatus`
- Modify `vdec.errorStatus`
- Modify `adec.stableOutput`
- Modify `vdec.stableOutput`
- Notify `OnDecodersStopped(*)`

Asynchronous Postcondition
- `ctrl.decodingState == uhAtscDecCtrl_StateStopped`
- `ctrl.trickState == uhAtscDec_TrickStateStopped`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— ctrl.pauseAtEnabled == False
— ctrl.pauseAtFirstFrame == False
— adec.errorStatus == uhAtscDecAudio_NoError
— vdec.errorStatus == uhAtscDecVideo_NoError
— adec.stableOutput == False
— vdec.stableOutput == False

Remarks
The error return value is present to prevent a race condition that would be present if a strong pre-condition were used. (The decoders can autonomously go to the Stopped state, in particular when a next I-frame has been decoded, see subclause 7.1.2.5.6.15.)

The error statuses of the audio decoder and video decoder are reset, but this triggers no OnRecovered notification(s).

7.1.2.5.4.8 GetDecodingState

Signature
```
uhErrorCode_t GetDecodingState (
    [out] puhAtscDecCtrl_State_t pDecState );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the operational state of the decoders.

Parameters

Table 153

Name	Description
pDecState	Output parameter returning the current decoder state.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pDecState == ctrl.decodingState

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.4.9 SetStreamMode

Signature
```
uhErrorCode_t SetStreamMode (
    [in] uhAtscDec_StreamMode_t streamMode );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the stream mode (push or pull) of the ATSC Decoder.

Parameters

Table 154

Name	Description
`streamMode`	The requested stream mode (push or pull).

Return Values
Standard.

Precondition
— `ctrl.demuxState == uhAtscDecCtrl_StateStopped`
— `ctrl.decodingState== uhAtscDecCtrl_StateStopped`

Action
— Modify `ctrl.streamMode`

Postcondition
— `ctrl.streamMode == streamMode`

Remarks
For details about the push and pull modes, see subclause 7.1.2.1.6.

7.1.2.5.4.10 GetStreamMode

Signature
```
uhErrorCode_t GetStreamMode (
    [out] puhAtscDec_StreamMode_t pStreamMode );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current stream mode.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 155

Name	Description
pStreamMode	Output parameter returning the current stream mode (push or pull).

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pStreamMode == ctrl.streamMode

7.1.2.5.5 uhIAtscDecStartStopNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in AtscDecCtrl related to the start/stop functionality. The Subscribe function of uhIAtscDecStartStop can be used to subscribe to these events.

Interface ID
uuid(57FD140F-4B84-438d-BD51-DB18E0291D0B)

7.1.2.5.5.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecCtrl_StartStopNtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.5.2 OnDemuxStarted

Signature

```
[async] Void OnDemuxStarted (
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `AtscDecCtrl` when the de-multiplexer is started.

Parameters

Table 156

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.5.3 OnDemuxStopped

Signature

```
[async] Void OnDemuxStopped (
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `AtscDecCtrl` when the de-multiplexer is stopped.

Parameters

Table 157

Name	Description
`cookie`	The value that was passed by the client at subscription time.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
See also subclause 7.1.2.3.4 for special execution constraints related to this notification.

7.1.2.5.5.4 OnDecodersStarted

Signature
```
[async] Void OnDecodersStarted (
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `AtscDecCtrl` when the decoders are started.

Parameters

Table 158

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.5.5 OnDecodersStopped

Signature

```
[async] Void OnDecodersStopped (
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `AtscDecCtrl` when the decoders are stopped.

Parameters

Table 159

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
See also subclause 7.1.2.3.4 for special execution constraints related to this notification.

7.1.2.5.6 uhlAtscDecTrick

Qualifiers
None.

Description
This interface of `AtscDecCtrl` provides functions to control trick play functionality. Note that this interface does not control trick play functionality for audio or video clip play.

Interface ID

```
uuid( 007099A0-3829-485D-A41E-2FC65B9CBE48 )
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.6.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAtscDecTrickNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecCtrl_TrickNtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.1.2.5.6.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAtscDecTrickNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecCtrl_TrickNtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

7.1.2.5.6.3 GetNrSuppSpeeds

Signature
```
uhErrorCode_t GetNrSuppSpeeds (
    [out] pUInt32 pNrOfSpeeds );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of supported speeds.

Parameters

Table 160

Name	Description
pNrOfSpeeds	Output parameter returning the number of supported speeds.

Return Values
Standard.

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
```
*pNrOfSpeeds == ctrl.nrSuppSpeeds
```

7.1.2.5.6.4 GetSuppSpeedSet

Signature
```
uhErrorCode_t GetSuppSpeedSet (
    [in] UInt32  nrOfSpeeds,
    [out, size_is(nrOfSpeeds)] puhAtscDec_Speed_t pSpeedSet );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the set of supported speeds.

Parameters

Table 161

Name	Description
`nrOfSpeeds`	The number of supported speeds as returned by the `GetNrSuppSpeeds` function.
`pSpeedSet`	Output parameter returning the set of supported speeds (as an array).

Return Values
Standard.

Precondition
```
nrOfSpeeds == ctrl.nrSuppSpeeds
```

Action
None.

Postcondition
```
*pSpeedSet == ctrl.suppSpeeds
```

7.1.2.5.6.5 SetSpeed

Signature
```
uhErrorCode_t SetSpeed (
    [in] uhAtscDec_Speed_t reqSpeed );
```

Qualifiers
— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Sets the decoding speed factor.

Parameters

Table 162

Name	Description
`reqSpeed`	The requested decoding speed factor.

Return Values
Standard.

Precondition
— `ctrl.streamMode == uhAtscDec_StreamModePull`
— `reqSpeed` in `ctrl.suppSpeeds`

Action
— Modify `ctrl.speed`

Postcondition
`ctrl.speed == reqSpeed`

7.1.2.5.6.6 GetSpeed

Signature
```
uhErrorCode_t GetSpeed (
    [out] puhAtscDec_Speed_t pReqSpeed );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the decoding speed factor.

Parameters

Table 163

Name	Description
`pReqSpeed`	Output parameter returning the decoding speed factor.

Return Values
Standard.

Precondition
`ctrl.streamMode == uhAtscDec_StreamModePull`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pReqSpeed == ctrl.speed
```

7.1.2.5.6.7 Pause

Signature
```
uhErrorCode_t Pause (
    [in] uhAtscDec_FrameType_t pauseType );
```

Qualifiers
— asynchronous

— single-threaded

Description
Pauses the decoding, taking into account the requested type of pause.

NOTE If `ctrl.trickState` changes to uhAtscDec`_TrickStateStopping` or uhAtscDec`_TrickStateStopped` before the execution of the asynchronous part of this function has started, the asynchronous part will not be executed. Always either all or nothing of the asynchronous part is executed.

Parameters

Table 164

Name	Description
`pauseType`	The type of pause (i.e. pause on next I-frame, next I- or P-frame, or any next frame).

Return Values
Only non-standard errors are listed here.

Table 165

Value	Description
`UH_ERR_ATSCDEC_PAUSE_IGNORED`	The request to pause has been ignored. This error is returned only if `ctrl.trickState == uhAtscDec_TrickStatePausing`

Precondition
— `ctrl.streamMode == uhAtscDec_StreamModePull`
— `(ctrl.trickState == uhAtscDec_TrickStateStarted)`
`|| (ctrl.trickState == uhAtscDec_TrickStatePausing)`

Action
— Modify `ctrl.trickState`

Postcondition
`ctrl.trickState == uhAtscDec_TrickStatePausing`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Asynchronous Action
— Modify `ctrl.trickState`
— Notify `OnPaused(*)`

Asynchronous Postcondition
`ctrl.trickState == uhAtscDec_TrickStatePaused`

7.1.2.5.6.8 SetPauseAt

Signature
```
uhErrorCode_t SetPauseAt(
    [in] uhTimeStamp_t pts,
    [in] uhAtscDec_FrameType_t pauseType );
```

Qualifiers
— synchronous

— single-threaded

Description
Requests that the decoding pauses at the given PTS, taking into account the requested type of pause.

Parameters

Table 166

Name	Description
`pts`	The PTS on which the pause is requested.
`pauseType`	The type of pause (i.e. pause on next I-frame, next I- or P-frame, or any next frame).

Return Values
Standard.

Precondition
— `ctrl.streamMode == uhAtscDec_StreamModePull`
— `ctrl.trickState == uhAtscDec_TrickStateStarted`
— `pts.timeStampHigh <= 1`

Action
— Modify `ctrl.pauseAtPts`
— Modify `ctrl.pauseAtType`
— Modify `ctrl.pauseAtEnabled`

Postcondition
— `ctrl.pauseAtPts == pts`
— `ctrl.pauseAtType == pauseType`
— `ctrl.pauseAtEnabled == True`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.6.9 GetPauseAt

Signature

```
uhErrorCode_t GetPauseAt (
    [out] pBool pPending,
    [out] puhTimeStamp_t pPts,
    [out] puhAtscDec_FrameType_t pPauseType );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the status of the PTS based pause.

Parameters

Table 167

Name	Description
`pPending`	Output parameter returning whether a pause at a certain PTS is pending.
`pPts`	Output parameter returning the PTS on which the pause is requested.
`pPauseType`	Output parameter returning the type of pause (i.e. pause on I-frame, I- or P-frame, or any frame).

Return Values

Standard.

Precondition

```
ctrl.streamMode == uhAtscDec_StreamModePull
```

Action

None.

Postcondition

— `*pPending == ctrl.pauseAtEnabled`
— `*pPts == ctrl.pauseAtPts`
— `*pPauseType == ctrl.pauseAtType`

7.1.2.5.6.10 CancelPauseAt

Signature

```
uhErrorCode_t CancelPauseAt ();
```

Qualifiers

— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Cancels the PTS based pause.

Parameters
None.

Return Values
Only non-standard errors are listed here.

Table 168

Value	Description
`UH_ERR_ATSCDEC_PAUSE_AT_NOT_ENABLED`	There is no “Pause At” request enabled at this moment.
`UH_ERR_ATSCDEC_CANCEL_IGNORED`	The request to cancel has been ignored, because the decoding is too close to the point of pausing for the pausing to be cancelled.

Precondition
`ctrl.streamMode == uhAtscDec_StreamModePull`

Action
— Modify `ctrl.pauseAtEnabled`

Postcondition
`ctrl.pauseAtEnabled == False`

Remarks
When the `UH_ERR_ATSCDEC_PAUSE_AT_NOT_ENABLED` error is returned, the ATSC decoder might just have paused the decoding because of a previous `PauseAt` request.

7.1.2.5.6.11 EnablePauseAtFirstFrame

Signature
```
uhErrorCode_t EnablePauseAtFirstFrame (
    [in] Bool firstFramePause );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables the pause at first frame when decoding will be started.

Parameters

Table 169

Name	Description
`firstFramePause`	The pause at first frame mode (enabled == `True`, disabled == `False`).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
— `ctrl.streamMode == uhAtscDec_StreamModePull`
— `ctrl.trickState == uhAtscDec_TrickStateStopped`

Action
— Modify `ctrl.pauseAtFirstFrame`

Postcondition
`ctrl.pauseAtFirstFrame == firstFramePause`

7.1.2.5.6.12 GetPauseAtFirstFrameEnabled

Signature
```
uhErrorCode_t GetPauseAtFirstFrameEnabled (
    [out] pBool pFirstFramePause );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the pause at first frame mode.

Parameters

Table 170

Name	Description
`pFirstFramePause`	Output parameter returning the pause at first frame mode (enabled == `True`, disabled == `False`).

Return Values
Standard.

Precondition
`ctrl.streamMode == uhAtscDec_StreamModePull`

Action
None.

Postcondition
`*pFirstFramePause == ctrl.pauseAtFirstFrame`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.6.13 Continue

Signature
```
uhErrorCode_t Continue ();
```

Qualifiers
- asynchronous
- single-threaded

Description
Continues decoding that was paused earlier.

NOTE If `ctrl.trickState` changes to uhAtscDec`_TrickStateStopping` or uhAtscDec`_TrickStateStopped` before the execution of the asynchronous part of this function has started, the asynchronous part will not be executed. Always either all or nothing of the asynchronous part is executed.

Parameters
None.

Return Values
Standard.

Precondition
- `ctrl.streamMode == uhAtscDec_StreamModePull`
- `ctrl.trickState == uhAtscDec_TrickStatePaused`

Action
- Modify `ctrl.trickState`

Postcondition
`ctrl.trickState == uhAtscDec_TrickStateContinuing`

Asynchronous Action
Modify ctrl.`trickState`

`Notify OnContinued(*)`

Asynchronous Postcondition
`ctrl.trickState == uhAtscDec_TrickStateStarted`

Remarks
The decoding continues with the last set speed.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.6.14 Step

Signature
```
uhErrorCode_t Step ();
```

Qualifiers
— asynchronous

— single-threaded

Description
Decodes the next frame and returns to pause. Depending on the frame decoding mode (`ctrl.frameDecMode`), it decodes the next I-frame, the next I- or P-frame or any next frame.

NOTE If `ctrl.trickState` changes to uhAtscDec_`TrickStateStopping` or uhAtscDec_`TrickStateStopped` before the execution of the asynchronous part of this function has started, the asynchronous part will not be executed. Always either all or nothing of the asynchronous part is executed.

Parameters
None.

Return Values
Standard.

Precondition
— `ctrl.streamMode == uhAtscDec_StreamModePull`
— `ctrl.trickState == uhAtscDec_TrickStatePaused`

Action
— Modify `ctrl.trickState`

Postcondition
```
ctrl.trickState == uhAtscDec_TrickStateStepping
```

Asynchronous Action
— Modify `ctrl.trickState`
— Notify `OnStepped(*)`

Asynchronous Postcondition
```
ctrl.trickState == uhAtscDec_TrickStatePaused
```

7.1.2.5.6.15 DecodeNextIframe

Signature
```
uhErrorCode_t DecodeNextIframe ();
```

Qualifiers
— asynchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Decodes the next I-frame (i.e. the first one encountered that is preceded by a sequence header) and returns to the stopped state. Any data still present in the video input buffer when the next I-frame has been decoded, may be lost. The client should not rely on this data.

While decoding the next I-frame, CC extraction is performed, but no audio is decoded.

NOTE If `ctrl.trickState` changes to uhAtscDec`_TrickStateStopping` or uhAtscDec`_TrickStateStopped` before the execution of the asynchronous part of this function has started, the asynchronous part will not be executed. Always either all or nothing of the asynchronous part is executed.

Parameters
None.

Return Values
Standard.

Precondition
— `ctrl.streamMode == uhAtscDec_StreamModePull`
— `ctrl.trickState == uhAtscDec_TrickStateStopped`

Action
— Modify `ctrl.trickState`
— Modify `ctrl.decodingState`

Postcondition
— `ctrl.trickState == uhAtscDec_TrickStateDecIframe`
— `ctrl.decodingState == uhAtscDecCtrl_StateStarted`

Asynchronous Action
— Modify `ctrl.trickState`
— Modify `ctrl.decodingState`
— Notify `OnNextIframeDecoded(*)`

Asynchronous Postcondition
— `ctrl.trickState == uhAtscDec_TrickStateStopped`
— `ctrl.decodingState == uhAtscDecCtrl_StateStopped`

Remarks
— Although this function changes `ctrl.decodingState`, it does not send `OnDecodersStarted` or `OnDecodersStopped` notifications.

— It is recommended that the client takes care that only the data related to a single I-frame is fed to the video decoder (through the video input buffer) each time it calls `DecodeNextIframe`. After the video decoder has decoded the I-frame, the client can take care that the next I-frame is fed to the video decoder for the next call to `DecodeNextIframe`.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.6.16 GetTrickState

Signature
```
uhErrorCode_t GetTrickState ([out] puhAtscDec_TrickState_t pTrickState );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current trick state.

Parameters

Table 171

Name	Description
pTrickState	Output parameter returning the current trick state.

Return Values
Standard.

Precondition
```
ctrl.streamMode == uhAtscDec_StreamModePull
```

Action
None.

Postcondition
```
*pTrickState == ctrl.trickState
```

7.1.2.5.6.17 GetSuppFrameDecModes

Signature
```
uhErrorCode_t GetSuppFrameDecMode(
    [out] puhAtscDec_FrameTypeSet_t pFrameDecModeSet);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the set of supported frame decoding modes (for pull mode).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 172

Name	Description
pFrameDecModeSet	Output parameter returning the set of supported frame decoding modes (I-frame, I- and P-frame, all frames).

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
*pFrameDecModeSet == ctrl.suppFrameDecModes
```

7.1.2.5.6.18 SetFrameDecMode

Signature
```
uhErrorCode_t SetFrameDecMode(
    [in] uhAtscDec_FrameType_t frameDecMode );
```

Qualifiers
- synchronous
- single-threaded

Description
Sets the frame decoding mode (for pull mode). In case of I-frame-only or I- and P-frame-only based decoding:

- The PTSs in the stream are ignored for timing the presentation of the video frames. The decoder decodes the frames as quickly as it can (at most one decoded frame per frame period). It may be the case that not every frame period a new frame is decoded.
- The input stream may be fragmented, e.g. frames may be removed from the input stream. The video decoder will scan for I-frames (and P-frames) in the input (by looking for the picture_start_code and checking the picture_coding_type) and decode these. Actual decoding starts with the first I-frame that is preceded by a sequence header.
- The currently set speed is ignored. Pause, PauseAt & Step are still allowed. The PauseAt pauses on a decoded frame that matches the “PauseAt PTS” or on a decoded frame that has a PTS that is later than the “PauseAt PTS” while the PTS of the frame decoded before that frame was earlier than the “PauseAt PTS”. Stepping steps to the next frame that is to be decoded given the currently set frame decoding mode.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 173

Name	Description
`frameDecMode`	The frame decoding mode (I-frame only, I- and P-frame-only, or all frames).

Return Values
Standard.

Precondition
— `ctrl.streamMode == uhAtscDec_StreamModePull`
— `ctrl.trickState == uhAtscDec_TrickStateStopped`
— `frameDecMode` in `ctrl.suppFrameDecModes`

Action
— Modify `ctrl.frameDecMode`

Postcondition
`ctrl.frameDecMode == frameDecMode`

Remarks
For I- and P-frame only decoding, use `uhAtscDec_IorPframe` (a frame is decoded when it is either an I-frame or a P-frame).

7.1.2.5.6.19 GetFrameDecMode

Signature
```
uhErrorCode_t GetFrameDecMode (
    [out] puhAtscDec_FrameType_t pFrameDecMode );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the frame decoding mode (for pull mode).

Parameters

Table 174

Name	Description
`pFrameDecMode`	Output parameter returning the frame decoding mode (I-frame only, I- and P-frame-only, or all frames).

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`ctrl.streamMode == uhAtscDec_StreamModePull`

Action
None.

Postcondition
`*pFrameDecMode == ctrl.frameDecMode`

7.1.2.5.7 uhIAtscDecTrickNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in `AtscDecCtrl` related to trick play functionality. The Subscribe function of `uhIAtscDecTrickNtf` can be used to subscribe to these events.

Interface ID
`uuid( AA1122F7-3F4B-4DF3-8FF0-E94E9A1446C1 )`

7.1.2.5.7.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecCtrl_TrickNtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

7.1.2.5.7.2 OnPaused

Signature
```
[async] Void OnPaused(
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `AtscDecCtrl` when the decoding has been paused.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 175

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.7.3 OnContinued

Signature

```
OnContinued(
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `AtscDecCtrl` when the decoding has been continued.

Parameters

Table 176

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.7.4 OnStepped

Signature
```
OnStepped(
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `AtscDecCtrl` when a decoding step has been made.

Parameters

Table 177

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.7.5 OnNextIframeDecoded

Signature
```
OnNextIframeDecoded(
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `AtscDecCtrl` when the next I-frame has been decoded.

Parameters

Table 178

Name	Description
`cookie`	The value that was passed by the client at subscription time.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.8 uhIAtscDecVideo

Qualifiers
None.

Description
This interface of `AtscDecVideo` provides functions to retrieve properties of the digital video input stream and provides a subscription and an unsubscription function on notifications of changes of these properties

Interface ID
`uuid( AFC001DE-4166-4685-AF58-FAFBC4464F3D )`

7.1.2.5.8.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAtscDecVideoNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecVideo_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.1.2.5.8.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAtscDecVideoNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecVideo_NtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.8.3 GetStandardData

Signature

```
uhErrorCode_t GetStandardData (
    [out] puhAtscDecVideo_CompStd_t              pStandard,
    [out] puhAtscDecVideo_Profile_t              pProfile,
    [out] puhAtscDecVideo_Level_t                pLevel );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the compression standard, MPEG-2 profile, and MPEG-2 level of the input stream.

Parameters

Table 179

Name	Description
`pStandard`	Output parameter returning the compression standard of the input stream.
`pProfile`	Output parameter returning the MPEG-2 profile of the input stream.
`pLevel`	Output parameter returning the MPEG-2 level of the input stream.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*pStandard == vdec.compressionStd`
— `*pProfile == vdec.profile`
— `*pLevel == vdec.level`

Remarks

When there is no video input stream,

— `*pStandard == uhAtscDecVideo_CompStdInvalid`
— `*pProfile == uhAtscDecVideo_InvalidProfile`
— `*pLevel == uhAtscDecVideo_LevelInvalid`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.8.4 GetPictureData

Signature

```
uhErrorCode_t GetPictureData (
     [out] puhAtscDec_ValStat_t        pStatus,
     [out] pUInt32                     pHeight,
     [out] pUInt32                     pWidth,
     [out] puhAtscDecVideo_FrameRate_t pFrameRate,
     [out] puhAtscDecVideo_ScanType_t  pScanType );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the resolution, its validity status, frame rate, and scan type of pictures in the input stream.

Parameters

Table 180

Name	Description
`pStatus`	Output parameter returning the validity status of the resolution (`vdec.height` and `vdec.width`) of the pictures in the input stream.
`pHeight`	Output parameter returning the height of pictures - in pixels - in the input stream.
`pWidth`	Output parameter returning the width of pictures - in pixels - in the input stream.
`pFrameRate`	Output parameter returning the frame rate of pictures in the input stream.
`pScanType`	Output parameter returning the scan type of pictures in the input stream.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*pStatus == vdec.resStatus`
— `*pHeight == vdec.height`
— `*pWidth == vdec.width`
— `*pFrameRate == vdec.frameRate`
— `*pScanType == vdec.scanType`

Remarks

When there is no input stream:

— Both `*pHeight` and `*pWidth` are `0`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— *pFrameRate == uhAtscDecVideo_InvalidFrameRate
— *pScanType == uhAtscDecVideo_ScanTypeInvalid

7.1.2.5.8.5 GetErrorStatus

Signature
```
uhErrorCode_t GetErrorStatus (
    [out] puhAtscDecVideo_ErrorSet_t pErrorStatus );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current error status.

Parameters

Table 181

Name	Description
pErrorStatus	Output parameter returning the current error status.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pErrorStatus == vdec.errorStatus

7.1.2.5.8.6 GetStableOutput

Signature
```
uhErrorCode_t GetStableOutput (
    [out] pBool pStableOutput);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the video decoder has currently stable output.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 182

Name	Description
pStableOutput	Output parameter returning whether the video decoder has currently stable output.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pStableOutput == vdec.stableOutput

7.1.2.5.9 uhIAtscDecVideoNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in the AtscDecVideo. The Subscribe function of uhIAtscDecVideo can be used to subscribe to these events.

Interface ID
uuid(0031DFE1-3114-4b7b-A905-47373169E4B6)

7.1.2.5.9.1 OnSubscriptionChanged

Signature
```
Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecVideo_NtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.9.2 OnStandardDataChanged

Signature

```
Void OnStandardDataChanged (
    [in] UInt32                    cookie,
    [in] uhAtscDecVideo_CompStd_t standard,
    [in] uhAtscDecVideo_Profile_t profile,
    [in] uhAtscDecVideo_Level_t   level );
```

Qualifiers
None.

Description
`AtscDecVideo` calls this function whenever it detects a change in the compression standard, profile, or level of its input stream.

Parameters

Table 183

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`standard`	The new compression standard of the input stream.
`profile`	The new profile of the input stream.
`level`	The new level of the input stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.9.3 OnPictureDataChanged

Signature

```
Void OnPictureDataChanged (
    [in] UInt32                     cookie,
    [in] uhAtscDec_ValStat_t        status,
    [in] UInt32                     height,
    [in] UInt32                     width,
    [in] uhAtscDecVideo_FrameRate_t frameRate,
    [in] uhAtscDecVideo_ScanType_t  scanType );
```

Qualifiers
None.

Description
`AtscDecVideo` calls this function whenever it detects a change in resolution, frame rate and scan type, of pictures in its input stream.

Parameters

Table 184

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the resolution (`vdec.height` and `vdec.width`) of the pictures in the input stream.
`height`	The new height of pictures - in pixels - in the input stream.
`width`	The new width of pictures - in pixels - in the input stream.
`frameRate`	The new frame rate of pictures in the input stream.
`scanType`	The new scan type of pictures in the input stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.9.4 OnError

Signature

```
Void OnError (
    [in] UInt32                       cookie,
    [in] uhAtscDecVideo_ErrorSet_t    errors );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that one or more errors occurred when decoding Video from the digital Video input stream.

Parameters

Table 185

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`errors`	Describes the type(s) of error(s) that occurred.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.9.5 OnRecovered

Signature

```
Void OnRecovered (
    [in] UInt32                       cookie,
    [in] uhAtscDecVideo_ErrorSet_t    errors );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that the video decoder has recovered from one or more previously notified errors.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 186

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`error`	Describes the type(s) of error(s) from which the video decoder has recovered.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.9.6 OnStableOutput

Signature

```
Void OnStableOutput (
    [in] UInt32 cookie,
    [in] Bool stableOutput );
```

Qualifiers
None.

Description
The `AtscDecVideo` calls this function when there is a change in the stable output of the `AtscDecVideo`.

Parameters

Table 187

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`stableOutput`	If `True`: `AtscDecVideo` is outputting decoded video long enough so that the client can safely unblank the corresponding output at the appropriate logical component(s) downstream. If `False`: `AtscDecVideo` is no longer outputting a stable (decoded) video stream.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.10 uhIAtscDecVideoUserData

Qualifiers
None.

Description
This interface of `AtscDecVideo` provides functions to subscribe and unsubscribe to the User Data detected as part of the video stream. This interface is optional and may not be supported by all decoders.

Interface ID
`uuid( 5D6867EB-8AE4-4909-A1A7-51E7D9C1C48B )`

7.1.2.5.10.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAtscDecVideoUserDataNtf *pIUserDataNotify,
    [in] UInt32 cookie,
    [in] uhAtscDecVideo_UserDataNtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.1.2.5.10.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAtscDecVideoUserDataNtf *pIUserDataNotify,
    [in] UInt32 cookie,
    [in] uhAtscDecVideo_UserDataNtfSet_t notifs);
```

Qualifiers
— unsubscribe-function

7.1.2.5.11 uhIAtscDecVideoUserDataNtf

Qualifiers
callback.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface of `AtscDecVideo` provides notifications on arrival of User Data as part of the video stream. The attributes of user data are provided as part of the notification.

Interface ID
`uuid( C65D3D89-9273-487e-ADBE-0D9AD51F29D1 )`

7.1.2.5.11.1 OnUserDataSubscriptionChanged

Signature
```
Void OnUserDataSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecVideo_UserDataNtfSet_t notifs );
```

Qualifiers
onsubscriptionchanged-function

7.1.2.5.11.2 OnUserDataSequenceArrived

Signature
```
Void OnUserDataSequenceArrived (

    [in] UInt32                      cookie,
    [in] uhAtscDec_ValStat_t         status,
    [in] uhTimeStamp_t               timeStamp,
    [in] UInt32                      size,
    [in, size_is(size)] const pUInt8 pUserData );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that sequence user data has arrived as part of the video stream. A pointer to the start of the buffer and valid number of bytes are passed as arguments. Along with the sequence user data, the corresponding incoming PTS is also sent to the client. Clients can use this PTS value to present any time-synchronised data present in `pUserData`.

Parameters

Table 188

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified sequence user data and timestamp.
`timestamp`	PTS associated with the user data.
`size`	Number of bytes of valid user data.
`pUserData`	Pointer to start of user data in memory.

Return Values
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
On return from the function call the data is no longer guaranteed to be available. The client is expected to copy the user data to its own memory area.

7.1.2.5.11.3 OnUserDataGopArrived

Signature

```
Void OnUserDataGopArrived (

    [in] UInt32                       cookie,
    [in] uhAtscDec_ValStat_t          status,
    [in] uhTimeStamp_t                timeStamp,
    [in] UInt32                       size,
    [in, size_is(size)] const pUInt8 pUserData );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that GOP user data has arrived as part of the video stream. A pointer to the start of the buffer and valid number of bytes are passed as arguments. Along with the GOP user data, the corresponding incoming PTS is also sent to the client. Clients can use this PTS value to present any time-synchronised data present in `pUserData`.

Parameters

Table 189

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified GOP user data and timestamp.
`timeStamp`	PTS associated with the user data.
`size`	Number of bytes of valid user data.
`pUserData`	Pointer to start of user data in memory.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
On return from the function call the data is no longer guaranteed to be available. The client is expected to copy the user data to its own memory area.

7.1.2.5.11.4 OnUserDataPictureArrived

Signature
```
Void OnUserDataPictureArrived (
    [in] UInt32                       cookie,
    [in] uhAtscDec_ValStat_t          status,
    [in] uhTimeStamp_t                timeStamp,
    [in] UInt32                       size,
    [in, size_is(size)] const pUInt8 pUserData );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that picture user data has arrived as part of the video stream. A pointer to the start of the buffer and the number of valid bytes are passed as arguments. Along with the Picture User Data, the corresponding PTS is also sent to the client. Clients can use this PTS value to present any time-synchronised data present in `pUserData`.

Parameters

Table 190

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified picture user data and timestamp.
`timeStamp`	PTS associated with the user data.
`size`	Number of bytes of valid user data.
`pUserData`	Pointer to start of user data in memory.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

Remarks
On return from the function call the data is no longer guaranteed to be available. The client is expected to copy the user data to its own memory area.

7.1.2.5.12 uhIAtscDecVideoClip

Qualifiers
None

Description
This interface of AtscDecVideo provides functions to command video clip play functionality and provides a subscribe and an unsubscribe function for events related to video clip play status.

Interface ID
`uuid( 864B60C1-BF54-4ABC-B3B9-67984E0A8399 )`

7.1.2.5.12.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAtscDecVideoClipNtf       *pINotify,
    [in] UInt32                       cookie,
    [in] uhAtscDecVideo_ClipNtfSet_t  notifs );
```

Qualifiers
— subscribe-function

7.1.2.5.12.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAtscDecVideoClipNtf      *pINotify,
    [in] UInt32                      cookie,
    [in] uhAtscDecVideo_ClipNtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.12.3 GetMaxNrBuffers

Signature

```
uhErrorCode_t GetMaxNrBuffers (
    [out] pUInt32 pMaxNrBuffers);
```

Qualifiers

— synchronous

— thread-safe

Description

This function queries how many buffers can be held by the video decoder at the same time for video clip play decoding.

Parameters

Table 191

Name	Description
pMaxNrBuffers	Output parameter returning the maximum number of buffers that can be held by the video decoder at the same time

Return Values

Standard.

Precondition

```
True
```

Action

None.

Postcondition

```
*pMaxNrBuffers == vdec.clipMaxNrBuffers
```

7.1.2.5.12.4 AddData

Signature

```
uhErrorCode_t AddData (
    [in] UInt32 id,
    [in] UInt32 size,
    [in, size_is(size)] const pUInt8 pData,
    [in] Bool endOfData );
```

Qualifiers

— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function adds a buffer pointed to by “pData”, identified by “id” containing “size” bytes of input data to be decoded. This function can also be called by the client(s) when video clip play has not (yet) been started.

Parameters

Table 192

Name	Description
`id`	Identification of the buffer. When the buffer is no longer needed by `uhI`AtscDecVideo, client will receive an “OnBufferAvailable” notification returning the “id”.
`size`	Size of the buffer pointed to by “pData”.
`pData`	Pointer to the buffer containing (a part of) the video clip to be decoded.
`endOfData`	Indicates whether the data currently added is the last byte of the currently added Clip Play. It is True if the data currently added is the last data of the currently added Clip Play. It is False if not. If this flag is True, the ATSC video decoder will notify to client(s) with an OnEndOfData notification when it decodes this buffer.

Return Values
Standard.

Precondition
— if `( endOfData == False )`
  — `pData != NULL`
  — `size > 0`
  — `vdec.clipNrOfCurrentBuffers < vdec.clipMaxNrBuffers`

Action
— Modify `vdec.avclip`

Postcondition
— A new buffer has been added to the set `vdec.avclip`. The fields of this buffer are initialized with the corresponding parameters of this function.

7.1.2.5.12.5 Start

Signature
```
uhErrorCode_t Start ( );
```

Qualifiers
— asynchronous

— single-threaded

Description
This function requests the ATSC video decoder to start video clip play decoding.

Parameters
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
— `vdec.clipCurrentState == uhAtscDecVideo_ClipStStopped`
— `ctrl.streamMode == uhAtscDec_StreamModePush`
— `ctrl.decodingState == uhAtscDecCtrl_StateStarted`

Action
— Request to start the video decoding for clip
— Modify `vdec.clipCurrentState`

Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStStarting`

Asynchronous Action
— Modify `vdec.clipCurrentState`
— Notify `OnStarted(*)`

Asynchronous Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStStarted`

Remarks

When clip play is started, the video decoder recovers from any errors from which it has not yet recovered. See also the Active Behavior in subclause 7.1.2.4.4.

7.1.2.5.12.6 Stop

Signature
`uhErrorCode_t Stop ( );`

Qualifiers
— asynchronous

— single-threaded

Description
This function requests the ATSC video decoder to stop video clip decoding. The video decoder resumes decoding the ATSC video stream coming from the demux.

Parameters
None.

Return Values
Standard.

Precondition
— `vdec.clipCurrentState != uhAtscDecVideo_ClipStStopping`
— `vdec.clipCurrentState != uhAtscDecVideo_ClipStStopped`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Request to stop the video decoding for clip
— Modify `vdec.clipCurrentState`

Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStStopping`

Asynchronous Action
— For each `b` in `vdec.avclip` (in the order they were added)

  — Notify `OnBufferAvailable(*)`

  — Modify `vdec.avclip`

  — Modify `vdec.clipCurrentState`

  — Notify `OnStopped(*)`

Asynchronous Postcondition
— `vdec.clipCurrentState == uhAtscDecVideo_ClipStStopped`
— `vdec.avclip == {}`

Remarks
When clip play is stopped, the video decoder recovers from any errors from which it has not yet recovered. See also the Active Behavior in subclause 7.1.2.4.4.

7.1.2.5.12.7 GetPauseSupp

Signature
```
uhErrorCode_t GetPauseSupp ([out] pBool pSupported );
```

Qualifiers
— synchronous

— thread-safe

Description
This function reports whether pausing for video clip decoding is supported by the ATSC video decoder.

Parameters

Table 193

Name	Description
`pSupported`	Output parameter reporting whether pausing is supported or not. It is `True` when pausing is supported; it is `False`, otherwise.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
`*pSupported == vdec.clipPauseSupported`

7.1.2.5.12.8 Pause

Signature
`uhErrorCode_t Pause ( );`

Qualifiers
— asynchronous

— single-threaded

Description
This function requests the ATSC video decoder to pause video clip decoding as soon as possible. The position where decoding is paused depends on the type of decoder and data being decoded.

Parameters
None.

Return Values
Standard.

Precondition
— `vdec.clipCurrentState == uhAtscDecVideo_ClipStStarted`
— `vdec.clipPauseSupported == True`

Action
— Request to pause the video decoding for clip
— Modify `vdec.clipCurrentState`

Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStPausing`

Asynchronous Action
— Modify `vdec.clipCurrentState`
— Notify `OnPaused(*)`

Asynchronous Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStPaused`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.12.9 Continue

Signature
```
uhErrorCode_t Continue ( );
```

Qualifiers
- asynchronous
- single-threaded

Description
This function requests the ATSC video decoder to continue video clip decoding after having been paused.

Parameters
None.

Return Values
Standard.

Precondition
- `vdec.clipCurrentState == uhAtscDecVideo_ClipStPaused`
- `vdec.clipPauseSupported == True`

Action
- Request to start the video decoding for clip
- Modify `vdec.clipCurrentState`

Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStStarting`

Asynchronous Action
- Modify `vdec.clipCurrentState`
- Notify `OnStarted(*)`

Asynchronous Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStStarted`

7.1.2.5.12.10 GetStepSupp

Signature
```
uhErrorCode_t GetStepSupp ([out] pBool pSupported );
```

Qualifiers
- synchronous
- thread-safe

Description
This function reports whether stepping for video clip is supported for ATSC video decoder for Video Clip Play.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 194

Name	Description
`pSupported`	Output parameter reporting whether stepping is supported or not for Video Clip Play. It is `True` when stepping is supported; it is `False`, otherwise.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pSupported == vdec.clipStepSupported`

7.1.2.5.12.11 Step

Signature
```
uhErrorCode_t Step ( );
```

Qualifiers
— asynchronous

— single-threaded

Description
This function requests the ATSC video decoder to decode the next video frame and then go in to the Paused state.

Parameters
None.

Return Values
Standard.

Precondition
— `vdec.clipCurrentState == uhAtscDecVideo_ClipStPaused`
— `vdec.clipStepSupported == True`

Action
— Request to decode the next frame of the video decoding for clip
— Modify `vdec.clipCurrentState`

Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStPausing`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Asynchronous Action
— Modify `vdec.clipCurrentState`
— Notify `OnPaused(*)`

Asynchronous Postcondition
`vdec.clipCurrentState == uhAtscDecVideo_ClipStPaused`

7.1.2.5.12.12 GetAvailableNrBuffers

Signature
```
uhErrorCode_t GetAvailableNrBuffers (
    [out] pUInt32 pNrBuffers);
```

Qualifiers
— synchronous

— thread-safe

Description
This function queries how many buffers are available for the client to add the video clip data.

Parameters

Table 195

Name	Description
`pNrBuffers`	Output parameter returning the available number of buffers that can be added.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pNrBuffers == vdec.clipMaxNrBuffers - vdec.clipNrOfCurrentBuffers`

7.1.2.5.12.13 GetState

Signature
```
uhErrorCode_t GetState (
    [out] puhAtscDecVideo_ClipState_t pClipState );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the current state of the video clip play.

Parameters

Table 196

Name	Description
`pClipState`	Output parameter returning the current state of the video clip play.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pClipState == vdec.clipCurrentState`

7.1.2.5.13 uhIAtscDecVideoClipNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that are related to decoding a video clip. The Subscribe function of `uhIAtscDecVideoClip` can be used to subscribe to these events.

Interface ID
`uuid( 2520BADC-5BB6-4905-BEAD-670AF59EC77B )`

7.1.2.5.13.1 OnSubscriptionChanged

Signature
```
Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecVideo_ClipNtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.13.2 OnBufferAvailable

Signature
```
Void OnBufferAvailable (
    [in] UInt32 cookie,
    [in] UInt32 id );
```

Qualifiers
None.

Description
This function is called by the ATSC Video Decoder whenever a buffer, which was filled with data, has become free. This may result from decoding of the data filled in this buffer, or from a request from the ATSC Video Decoder Client to stop the decoding of the video clip.

The buffers are becoming free for further addition of new data, and notified, in the same order as they were added by the ATSC Video Decoder Client.

Parameters

Table 197

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`id`	Identification for the buffer provided by the client.

Return Values
None.

Precondition
`True`

Action
— Any callback-complaint action.

Postcondition
`True`

Remarks
Each call of `AddData()` with a non-empty buffer eventually results in an `OnBufferAvailable` event.

7.1.2.5.13.3 OnEndOfData

Signature
```
Void OnEndOfData (
    [in] UInt32 cookie );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is called by the ATSC video decoder when it has completed the decoding of a client buffer which has the "`endOfData`" flag set or when it has encountered a client buffer which has the "`endOfData`" flag set with the size of this buffer being 0.

Parameters

Table 198

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-complaint action

Postcondition
`True`

7.1.2.5.13.4 OnStarted

Signature
```
Void OnStarted (
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the ATSC video decoder if video clip play has started or continued as a result of the "`Start`" or "`Continue`" function.

Parameters

Table 199

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.13.5 OnStopped

Signature
```
Void OnStopped (
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the ATSC video decoder if video clip play has stopped as a result of the "`Stop`" function.

Parameters

Table 200

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.13.6 OnPaused

Signature
```
Void OnPaused (
    [in] UInt32 cookie );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is called by the ATSC video decoder if video clip play has been paused as a result of either the "`Pause`" or "`Step`"function.

Parameters

Table 201

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.14 uhIAtscDecCcExtr

Qualifiers
None.

Description
This interface of `AtscDecCcExtr` provides facilities to subscribe/unsubscribe to notifications of events related to the extraction of CC data. It also provides facilities to select specific CC services for which EIA-708 CC Service Blocks (see clause 6 of [15]) will be notified to `AtscDecClient`, if subscribed to the appropriate notification.

Interface ID
`uuid( E263C0A1-2792-489f-BA11-825A424A98AA )`

7.1.2.5.14.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAtscDecCcExtrNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecCcExtr_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.14.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIAtscDecCcExtrNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecCcExtr_NtfSet_t notifs );
```

Qualifiers

— unsubscribe-function

7.1.2.5.14.3 GetMaxNrServices

Signature

```
uhErrorCode_t GetMaxNrServices ( [out] pUInt32 pMaxNrServices );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the maximum number of close caption services that can be selected.

Parameters

Table 202

Name	Description
`pMaxNrServices`	Output parameter returning the maximum number of close caption services for which CC Service Blocks can be notified via `OnSelectedServiceData`.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— **pMaxNrServices* `== ccextr.maxNrServices`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.14.4 SetServices

Signature

```
uhErrorCode_t SetServices (
    [in] UInt32 nrServices,
    [in, size_is(nrServices)] pUInt32 pServices );
```

Qualifiers

— synchronous

— thread-safe

Description

Sets the list with service numbers of close caption services, for which CC Service Blocks are be notified via `OnSelectedServiceData`.

Parameters

Table 203

Name	Description
`nrServices`	The number of elements, which are passed in `pServices`.
`pServices`	Pointer to an array with close caption services for which CC Service Blocks are notified via `OnSelectedServiceData`. The close caption services are identified by their service number.

Return Values

Standard.

Precondition

— `nrServices ≤ ccextr.maxNrServices`

Action

— `Modify ccextr.nrServicesSelected`
— for each `i in {0 .. nrServices-1}`
  — `Modify ccextr.servicesSelected[i]`
— for each `i in {nrServices .. ccextr.maxNrServices-1}`
  — `Reset ccextr.servicesSelected[i]`

Postcondition

— `ccextr.nrServicesSelected == nrServices`
— for each `i in {0 .. nrServices-1}`
  — `ccextr.servicesSelected[i]== pServices[i]`
— for each `i in {nrServices .. ccextr.maxNrServices-1}`
  — `ccextr.servicesSelected[i] == 0`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.14.5 GetServices

Signature

```
uhErrorCode_t GetServices (
    [in] UInt32 nrRequestedServices,
    [out] pUInt32 pNrServices,
    [out, size_is(nrRequestedServices)] pUInt32 pServices);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the list with service numbers of close caption services, for which CC Service Blocks are notified via `OnSelectedServiceData`.

Parameters

Table 204

Name	Description
`nrRequestedServices`	The number of elements, which is allocated space for in `pServices`.
`pNrServices`	The total number of closed caption services for which CC Services Blocks are notified.
`pServices`	Pointer to an array with close caption services for which CC Service Blocks are notified via `OnSelectedServiceData`. The services are identified by their service number. The number of services returned in the array, is the minimum between `nrRequestedServices` and `ccextr.nrServicesSelected`.

Return Values

Standard.

Precondition

— `nrRequestedServices <= ccextr.maxNrServices`

Action

None.

Postcondition

— `*pNrServices == ccextr.nrServicesSelected`
— if (`nrRequestedServices <= ccextr.nrServicesSelected`)
  — for each `i in {0 .. nrRequestedServices-1}`
    — `pServices[i] == ccextr.servicesSelected[i]`
— else
  — for each `i in {0 .. ccextr.nrServicesSelected-1}`
    — `pServices[i] == ccextr.servicesSelected[i]`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.15 uhIAtscDecCcExtrNtf

Qualifiers
— callback

Description
This interface should be provided by each AtscDecCcExtr client that wants to subscribe to notifications of events related to closed caption data extraction that occurs in the AtscDecCcExtr. The Subscribe function of uhIAtscDecCcExtr can be used to subscribe to these events.

Interface ID
uuid(55202F1C-CBEF-4c41-BACF-CE5C0C7200F0)

7.1.2.5.15.1 OnSubscriptionChanged

Signature
```
Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecCcExtr_NtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

7.1.2.5.15.2 OnEia708Data

Signature
```
Void OnEia708Data (
    [in] UInt32                       cookie,
    [in] uhAtscDec_ValStat_t          status,
    [in] Bool                         ptsPresent,
    [in] uhTimeStamp_t                timeStamp,
    [in] UInt32                       size,
    [in, size_is(size)] const pUInt8 pData );
```

Qualifiers
None.

Description
AtscDecCcExtr calls this function whenever it has detected the end of an EIA-708 digital Caption Channel Packet in the user data input stream. *pData* points to the CC data. The CC Extractor owns the storage space occupied by the data.

First, the CC Extractor has detected the presence of EIA-708 data in the user data input stream, it has detected the start header of an EIA-708 Caption Channel Packet, and it has extracted the related data from the incoming stream and buffered it internally in the appropriate order for further decoding (see subclause 4.4.1 of [15]).

The storage space, and therefore the CC data, is guaranteed to stay unmodified until the current call of this function returns.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 205

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified EIA-708 CC data and related properties notified by this function.
`ptsPresent`	Value that indicates whether *timeStamp* contains a valid PTS. Is `True` if valid.
`timeStamp`	The presentation time stamp of the last Caption Channel Packet in *pData*. If *ptsPresent* is `False`, the content of `timeStamp` is irrelevant.
`size`	The size, in bytes, of valid DTVCC data to which `pData` points.
`pData`	Pointer to a size bytes buffer of valid DTVCC data.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
— DTVCC data (compliant with EIA-708) is encoded in variable-rate and variable-sized Caption Channel Packets; see subclause 4.4.1 of [15] for more details about Caption Channel Packets. The first two bytes of a Caption Channel Packet have the “Channel Packet Start” type. The remaining bytes of the Caption Channel Packet have the “Channel Packet Data” type. In the Picture User Data bits the end of a Caption Channel Packet is indicated by either:

  — extraction of a Caption Channel Packet with the “Channel Packet Start” type;

  — extraction of a Caption Channel Packet with the “Channel Packet Data” type associated with a `ccValid` flag set to False; see [15].

— An example of the organization of the storage space is shown in Figure 26:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

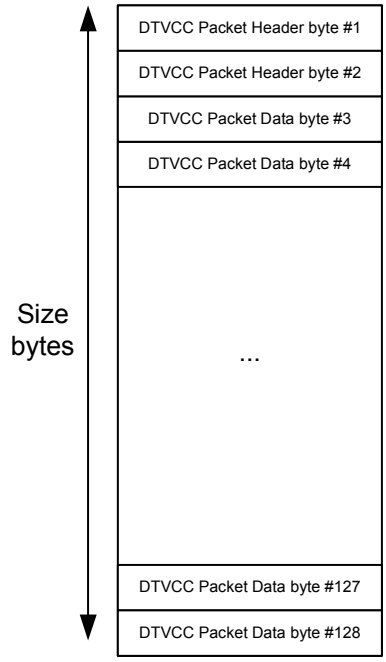


Figure 26 — Organization of capture DTVCC data, when a DTVCC Caption Channel Packet is 128 bytes in length. The first two bytes are the header of the DTVCC Caption Channel Packet.

7.1.2.5.15.3 OnSelectedServiceData

Signature

```
Void OnSelectedServiceData (
    [in] UInt32                      cookie,
    [in] uhAtscDec_ValStat_t         status,
    [in] Bool                        ptsPresent,
    [in] uhTimeStamp_t               timeStamp,
    [in] UInt32                      size,
    [in, size_is(size)] const pUInt8 pData,
    [in] UInt32                      service );
```

Qualifiers
None.

Description
This function is called by the Closed Caption Extractor whenever it has extracted a CC Service Block of the Caption Service with Service Number service, which belongs to the set of selected services. See clause 6 of [15] for the CC Service Blocks syntax, the data is notified in the appropriate order for further decoding; see subclause 4.4.1 of [15].

Parameters

Table 206

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified CC Service Block and related properties notified by this function.
`ptsPresent`	Value that indicates whether *timeStamp* contains a valid PTS. Is `True` if valid.

-`,,`,,`,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
timeStamp	The presentation time stamp of the CC data in *pData*. If *ptsPresent* is False, the content of *timeStamp* is irrelevant.
size	The size, in bytes, of valid CC data of the closed caption Service with Service Number service to which *pData* points.
pData	Pointer to a size bytes buffer of valid CC data of the closed caption Service with Service Number service.
service	The Service Number of the closed caption Service from which the CC Service Block in *pData* is extracted.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

7.1.2.5.15.4 OnEia608Data

Signature
```
Void OnEia608Data (
    [in] UInt32                                    cookie,
    [in] uhAtscDec_ValStat_t                       status,
    [in] UInt32                                    nrEia608Data,
    [in] Bool                                      ptsPresent,
    [in, size_is(nrEia608Data)] const puhTimeStamp_t pTimeStamp,
    [in, size_is(nrEia608Data)] const pBool        pField,
    [in, size_is(nrEia608Data)] const pUInt16      pData );
```

Qualifiers
None.

Description
This function is called by the Closed Caption Extractor whenever it has extracted analog CC data (NTSC line 21, odd and/or even field) from the user data input stream (see [15]). The bit layout of the notified analog CC data follows the CC standard (see [14]); this is shown in Figure 27.

Parameters

Table 207

Name	Description
cookie	The value that was passed by the client at subscription time.
status	The validity status of the notified EIA-608 CC data and related properties notified by this function.
nrEia608Data	The number of byte-pairs of valid analog CC data to which *pData* points.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
`ptsPresent`	Value that indicates whether *pTimeStamp* contains valid PTS. Is `True` if valid.
`pTimeStamp`	Pointer to a buffer containing the presentation time stamps of the analog CC data to which *pData* points. *pTimeStamp*`[i]` contains the presentation time stamp of the byte-pair of analog CC data in *pData*`[i]`. If *ptsPresent* is `False`, the content of *pTimeStamp* is irrelevant.
`pField`	Pointer to a buffer containing for each byte-pair of analog CC data in *pData* the indication of the field to which this byte-pair of analog CC data belongs. *pField*`[i]` contains the indication of the field to which the byte-pair of analog CC data in *pData*`[i]` belongs. It is `True` for the even field and `False` for the odd field.
`pData`	Pointer to a buffer of `nrEia608Data` byte-pairs of valid analog CC data extracted from the user data input stream. Each byte-pairs are stored as indicated in Figure 27.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
In case, NTSC line 21 CC data (see [15]) is present in the user data input stream of the Closed Caption Extractor, the CC data is transmitted in the decode order. The bit layout of the notified analog CC data in *pData* follows the CC standard (see [14]); this is shown in Figure 27.

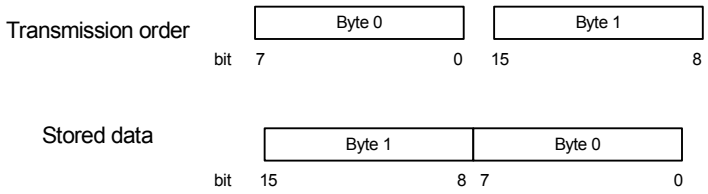


Figure 27 — The relationship of the bits layout of the captured CC data in *pData* and the order of transmission of CC data in case NTSC Line 21 closed caption data is present in the user data input stream.

7.1.2.5.16 uhlAtscDecAudio

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface of `AtscDecAudio` provides functions to retrieve properties of the digital audio input stream (or the audio clip input) and provides a subscription and an unsubscription function on notifications of changes of these properties.

Interface ID
```
uuid( 98AE108A-CCEE-42B0-AA9E-7725F0D721EB )
```

Execution Constraints
None.

7.1.2.5.16.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAtscDecAudioNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.1.2.5.16.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAtscDecAudioNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_NtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

7.1.2.5.16.3 GetStandard

Signature
```
uhErrorCode_t GetStandard ( [out] puhAtscDecAudio_CompStd_t pStandard );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the compression standard of the digital audio input stream.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 208

Name	Description
`pStandard`	Output parameter returning the compression standard of the digital audio input stream of the Decoder.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— *`*pStandard`* `== adec.compStandard`

Remarks
When there is no audio input stream in the Decoder, *`*pStandard`* `== uhAtscDecAudio_CompStdInvalid`.

7.1.2.5.16.4 GetSampFreq

Signature
```
uhErrorCode_t GetSampFreq ( [out] puhAtscDecAudio_SampFreq_t pSampFreq );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the sampling frequency of the digital audio input stream.

Parameters

Table 209

Name	Description
`pSampFreq`	Output parameter returning the sampling frequency of the digital audio input stream.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— `*pSampFreq == adec.sampFreq`

7.1.2.5.16.5 GetBitRate

Signature
```
uhErrorCode_t GetBitRate (
    [out] puhAtscDec_ValStat_t pStatus,
    [out] pBool                pVariable,
    [out] pUInt32              pBitRate );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the bit rate of the digital audio input stream is variable or constant. Also returns the bit rate (in kbit/s) of the digital audio input stream. When the input stream has a variable bit rate or when the bit rate of the input stream is unknown, the returned bit rate is 0.

Parameters

Table 210

Name	Description
`pStatus`	Output parameter returning the validity status of the bit rate properties (`adec.variableBitRate`, `adec.bitRate`) of the digital audio input stream.
`pVariable`	Output parameter returning whether the bit rate of the digital audio input stream is a variable bit rate (VBR). It is `True` when the bit rate of the digital audio input stream is variable. It is `False` when the bit rate of the digital audio input stream is constant.
`pBitRate`	Output parameter returning the bit rate (in kbit/s) of the digital audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pStatus == adec.bitRateStatus;`
— `*pVariable == adec.variableBitRate`
— `*pBitRate == adec.bitRate`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.16.6 GetChannels

Signature

```
uhErrorCode_t GetChannels (
    [out] puhAtscDec_ValStat_t pStatus,
    [out] pUInt32              pFrontChannels,
    [out] pUInt32              pRearChannels,
    [out] pBool                pLfe );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the number of front channels and rear channels in the digital audio input stream. Also returns whether a LFE channel is present in the digital audio input stream.

Parameters

Table 211

Name	Description
`pStatus`	Output parameter returning the validity status of the audio channel configuration (`adec.frontChannels`, `adec.rearChannels`, `adec.lfeChannelPresent`).
`pFrontChannels`	Output parameter returning the number of front channels in the digital audio input stream.
`pRearChannels`	Output parameter returning the number of rear channels in the digital audio input stream.
`pLfe`	Output parameter returning whether a LFE channel is present in the digital audio input stream. Is `True` if a LFE channel is present in the digital audio input stream, and `False` if not.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*pStatus == adec.channelStatus`
— `*pFrontChannels == adec.frontChannels`
— `*pRearChannels == adec.rearChannels`
— `*pLfe == adec.lfeChannelPresent`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.16.7 GetServiceType

Signature
```
uhErrorCode_t GetServiceType (
    [out] puhAtscDecAudio_ServiceType_t pServiceType );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the type of the audio service that the digital audio input stream conveys.

Parameters

Table 212

Name	Description
`pServiceType`	Output parameter returning the type of the audio service that the digital audio input stream conveys.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*`*`pServiceType`* `== adec.serviceType`

7.1.2.5.16.8 GetDolbySurroundInfo

Signature
```
uhErrorCode_t GetDolbySurroundInfo (
    [out] puhAtscDecAudio_DolbySurroundInfo_t pDolbySurround );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether Dolby Surround is present in the digital audio input stream.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 213

Name	Description
pDolbySurround	Output parameter returning whether Dolby Surround is present in the digital audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— **pDolbySurround* `==` `adec.dolbySurroundPresent`

7.1.2.5.16.9 GetErrorStatus

Signature
```
uhErrorCode_t GetErrorStatus (
    [out] puhAtscDecAudio_ErrorSet_t pErrorStatus );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current error status.

Parameters

Table 214

Name	Description
pErrorStatus	Output parameter returning the current error status.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pErrorStatus == adec.errorStatus`

7.1.2.5.16.10 GetStableOutput

Signature
```
uhErrorCode_t GetStableOutput (
    [out] pBool pStableOutput);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the audio decoder has currently stable output.

Parameters

Table 215

Name	Description
`pStableOutput`	Output parameter returning whether the audio decoder has currently stable output.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pStableOutput == adec.stableOutput`

7.1.2.5.17 uhIAtscDecAudioNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in `AtscDecAudio`. The Subscribe function of `uhIAtscDecAudio` can be used to subscribe to these events.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 436E902F-A279-4e13-AB74-58B5CF90E1EB )`

Execution Constraints
None.

7.1.2.5.17.1 OnSubscriptionChanged

Signature
```
Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_NtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

7.1.2.5.17.2 OnStandardChanged

Signature
```
Void OnStandardChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_CompStd_t standard );
```

Qualifiers
None.

Description
This function is called by `AtscDecAudio` whenever it detects a change in the compression standard of its digital audio input stream.

Parameters

Table 216

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`standard`	The new compression standard of the digital audio input stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.17.3 OnSampFreqChanged

Signature
```
Void OnSampFreqChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_SampFreq_t sampFreq );
```

Qualifiers
None.

Description
This function is called by `AtscDecAudio` whenever it detects a change in the sampling frequency of its digital audio input stream.

Parameters

Table 217

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`sampFreq`	The new sampling frequency detected by the Decoder.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.17.4 OnBitRateChanged

Signature
```
Void OnBitRateChanged (
    [in] UInt32 cookie,
    [in] uhAtscDec_ValStat_t status,
    [in] Bool variable,
    [in] UInt32 bitRate );
```

Qualifiers
None.

Description
This function is called by `AtscDecAudio` whenever it detects a change in the bit rate of its digital audio input stream and/or in the variability of the bit rate.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 218

Name	Description
cookie	The value that was passed by the client at subscription time.
status	The validity status of the notified bit rate properties (adec.variableBitRate, adec.bitRate).
variable	The new value indicating whether the bit rate of the digital audio input stream is variable. It is True if the bit rate of the digital audio input stream is variable and it is False if the bit rate of the digital audio input stream is constant.
bitRate	The new bit rate (in kbit/s) detected by the Decoder. When the digital audio input stream has a variable bit rate or when the bit rate of the digital audio input stream is unknown bitRate is 0.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

7.1.2.5.17.5 OnChannelsChanged

Signature

```
Void OnChannelsChanged (
    [in] UInt32              cookie,
    [in] uhAtscDec_ValStat_t status,
    [in] UInt32              frontChannels,
    [in] UInt32              rearChannels,
    [in] Bool                lfe );
```

Qualifiers
None.

Description
This function is called by AtscDecAudio whenever it detects a change in the configuration of the channels in its digital audio input stream.

Parameters

Table 219

Name	Description
cookie	The value that was passed by the client at subscription time.
status	The validity status of the notified audio channel configuration (adec.frontChannels, adec.rearChannels, adec.lfeChannelPresent).
frontChannels	The new number of front channels detected by the Decoder.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
rearChannels	The new number of rear channels detected by the Decoder.
lfe	The new value indicating the presence of a LFE channel in the digital audio input stream. Is True if a LFE channel is present in the digital audio input stream, is False if a LFE channel is not present in the digital audio input stream.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

7.1.2.5.17.6 OnServiceTypeChanged

Signature
```
Void OnServiceTypeChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_ServiceType_t serviceType );
```

Qualifiers
None.

Description
AtscDecAudio calls this function whenever it detects a change in the type of audio service that the digital audio input stream conveys.

Parameters

Table 220

Name	Description
cookie	The value that was passed by the client at subscription time.
serviceType	The new value of the type of audio service detected by the Decoder.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

7.1.2.5.17.7 OnDolbySurroundChanged

Signature
```
Void OnDolbySurroundChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_DolbySurroundInfo_t dolbySurround );
```

Qualifiers
None.

Description
This function is called by `AtscDecAudio` whenever it detects a change in the presence of Dolby Surround in its digital audio input stream.

Parameters

Table 221

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`dolbySurround`	The new value indicating the presence of Dolby Surround in the digital audio input stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.17.8 OnError

Signature
```
Void OnError (
    [in] UInt32                    cookie,
    [in] uhAtscDecAudio_ErrorSet_t errors );
```

Qualifiers
None.

Description
This notification function is used by `AtscDecAudio` to inform the subscribed client that one or more errors occurred when decoding audio from the digital audio input stream.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 222

Name	Description
cookie	The value that was passed by the client at subscription time.
errors	Describes the type(s) of error(s) that occurred.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

7.1.2.5.17.9 OnRecovered

Signature

```
Void OnRecovered (
    [in] UInt32                    cookie,
    [in] uhAtscDecAudio_ErrorSet_t errors);
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that the audio decoder has recovered from one or more previously notified errors.

Parameters

Table 223

Name	Description
cookie	The value that was passed by the client at subscription time.
errors	Describes the type(s) of error(s) from which the audio decoder has recovered.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

7.1.2.5.17.10 OnStableOutput

Signature
```
Void OnStableOutput (
    [in] UInt32 cookie,
    [in] Bool stableOutput );
```

Qualifiers
None.

Description
The `AtscDecAudio` calls this function when there is a change in the stable output of the `AtscDecAudio`.

Parameters

Table 224

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`stableOutput`	If `True`: `AtscDecAudio` is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream. If `False`: `AtscDecAudio` is no longer outputting a stable (decoded) audio stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.2.5.18 uhlAtscDecAudioClip

Qualifiers
None

Description
This interface of AtscDecAudio provides functions to command Audio clip play functionality and provides a subscribe and an unsubscribe function for events related to clip play status.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
uuid(D8B397C9-F2C6-424C-B767-7B7EE61D97C9)

7.1.2.5.18.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAtscDecAudioClipNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_ClipNtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.1.2.5.18.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAtscDecAudioClipNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_ClipNtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

7.1.2.5.18.3 GetMaxNrBuffers

Signature
```
uhErrorCode_t GetMaxNrBuffers (
    [out] pUInt32 pMaxNrBuffers );
```

Qualifiers
— synchronous

— thread-safe

Description
This function queries how many buffers can be held by the audio decoder at the same time for audio clip paly decoding.

Parameters

Table 225

Name	Description
pMaxNrBuffers	Output parameter returning the maximum number of buffers that can held by the audio decoder at the same time.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
`*pMaxNrBuffers == adec.clipMaxNrBuffers`

7.1.2.5.18.4 AddData

Signature
```
uhErrorCode_t AddData (
    [in] UInt32 id,
    [in] UInt32 size,
    [in, size_is(size)] const pUInt8 pData,
    [in] Bool endOfData );
```

Qualifiers
- synchronous
- single-threaded

Description
This function adds a buffer pointed by "`pData`", identified by "`id`" containing "`size`" bytes of input data to be decoded. This function can be called by the client(s) before switching into audio clip play decoding.

Parameters

Table 226

Name	Description
`id`	Identification of the buffer. When the buffer is no longer needed by the audio decoder, the client will receive an "`OnBufferAvailable`" notification returning the "`id`".
`size`	Size of the buffer pointed to by "`pData`".
`pData`	Pointer to the buffer containing (a part of) the Audio clip to be decoded.
`endOfData`	Indicates whether the data currently added is the last byte of the currently added Clip Play. It is `True` if the data currently added is the last data of the currently added Clip Play. It is `False` if not. If this flag is `True`, the ATSC Audio Decoder will notify to client(s) with an `OnEndOfData` notification when it decodes this buffer

Return Values
Standard.

Precondition
- `if ( endOfData == False )`
  - `pData != NULL`
  - `size >0`
  - `adec.clipNrOfCurrentBuffers < adec.clipMaxNrBuffers`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `adec.avclip`

Postcondition
— A new buffer has been added to the set `adec.avclip`. The fields of this buffer are initialized with the corresponding parameters of this function.

7.1.2.5.18.5 Start

Signature
```
uhErrorCode_t Start ( );
```

Qualifiers
— asynchronous

— single-threaded

Description
This function requests the ATSC Audio decoder to start audio clip play decoding.

Parameters
None.

Return Values
Standard.

Precondition
— `adec.clipCurrentState == uhAtscDecAudio_ClipStStopped`
— `ctrl.streamMode == uhAtscDec_StreamModePush`
— `ctrl.decodingState == uhAtscDecCtrl_StateStarted`

Action
— Request to start the audio decoding for clip
— Modify `adec.clipCurrentState`

Postcondition
`adec.clipCurrentState == uhAtscDecAudio_ClipStStarting`

Asynchronous Action
— Modify `adec.clipCurrentState`
— Notify `OnStarted(*)`

Asynchronous Postcondition
`adec.clipCurrentState == uhAtscDecAudio_ClipStStarted`

Remarks
— When clip play is started, the audio decoder recovers from any errors from which it has not yet recovered. See also the Active Behavior in subclause 7.1.2.4.5.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.18.6 Stop

Signature
`uhErrorCode_t Stop ( );`

Qualifiers
— asynchronous

— single-threaded

Description
This function requests the ATSC Audio Decoder to stop audio clip decoding. The audio decoder resumes decoding the ATSC audio stream coming from the demux.

Parameters
None.

Return Values
Standard.

Precondition
— `adec.clipCurrentState != uhAtscDecAudio_ClipStStopping`
— `adec.clipCurrentState != uhAtscDecAudio_ClipStStopped`

Action
— Request to stop the audio decoding for clip
— Modify `adec.clipCurrentState`

Postcondition
`adec.clipCurrentState == uhAtscDecAudio_ClipStStopping`

Asynchronous Action
— For each `b` in `adec.avclip` (in the order they were added)
  — Notify `OnBufferAvailable(*)`
  — Modify `adec.avclip`
  — Modify `adec.clipCurrentState`
  — Notify `OnStopped(*)`

Asynchronous Postcondition
`adec.clipCurrentState == uhAtscDecAudio_ClipStStopped`

`vdec.avclip == {}`

Remarks

— When clip play is stopped, the audio decoder recovers from any errors from which it has not yet recovered. See also the Active Behavior in subclause 7.1.2.4.5.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.18.7 GetPauseSupp

Signature
```
uhErrorCode_t GetPauseSupp ( [out] pBool pSupported );
```

Qualifiers
— synchronous

— thread-safe

Description
This function reports whether pausing for Audio clip decoding is supported by the ATSC Audio decoder.

Parameters

Table 227

Name	Description
`pSupported`	Output parameter reporting whether pausing is supported or not. It is `True` when pausing is supported; it is `False`, otherwise.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
```
*pSupported   == adec.clipPauseSupported
```

7.1.2.5.18.8 Pause

Signature
```
uhErrorCode_t Pause ( );
```

Qualifiers
— asynchronous

— single-threaded

Description
This function requests the ATSC Audio decoder to pause audio clip decoding as soon as possible. The position where decoding is paused depends on the type of decoder and data being decoded.

Parameters
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
— `adec.clipCurrentState == uhAtscDecAudio_ClipStStarted`
— `adec.clipPauseSupported == True`

Action
— Request to pause the audio decoding for clip
— Modify `adec.clipCurrentState`

Postcondition
`adec.clipCurrentState == uhAtscDecAudio_ClipStPausing`

Asynchronous Action
— Modify `adec.clipCurrentState`
— Notify `OnPaused(*)`

Asynchronous Postcondition
`adec.clipCurrentState == uhAtscDecAudio_ClipStPaused`

7.1.2.5.18.9 Continue

Signature
`uhErrorCode_t Continue( );`

Qualifiers
— asynchronous

— single-threaded

Description
This function requests the ATSC Audio decoder to continue audio clip decoding after having been paused.

Parameters
None.

Return Values
Standard.

Precondition
— `adec.clipCurrentState == uhAtscDecAudio_ClipStPaused`
— `adec.clipPauseSupported == True`

Action
— Request to start the audio decoding for clip
— Modify `adec.clipCurrentState`

Postcondition
`adec.clipCurrentState == uhAtscDecAudio_ClipStStarting`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Asynchronous Action
— Modify `adec.clipCurrentState`
— Notify `OnStarted(*)`

Asynchronous Postcondition
`adec.clipCurrentState == uhAtscDecAudio_ClipStStarted`

7.1.2.5.18.10 GetAvailableNrBuffers

Signature
```
uhErrorCode_t GetAvailableNrBuffers (
    [out] pUInt32 pNrBuffers );
```

Qualifiers
— synchronous

— thread-safe

Description
This function queries how many buffers are available for the client to add the audio clip data.

Parameters

Table 228

Name	Description
`pNrBuffers`	Output parameter returning the available number of buffers that can be added.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pNrBuffers == adec.clipMaxNrBuffers – adec.clipNrOfCurrentBuffers`

7.1.2.5.18.11 GetState

Signature
```
uhErrorCode_t GetState ( [out] puhAtscDecAudio_ClipState_t pClipState );
```

Qualifiers
— synchronous

— thread-safe

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the current state of the audio clip play.

Parameters

Table 229

Name	Description
`pClipState`	Output parameter returning the current state of the audio clip play.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pClipState == adec.clipCurrentState`

7.1.2.5.19 uhIAtscDecAudioClipNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that are related to decoding an audio clip. The Subscribe function of `uhI`AtscDecAudioClip can be used to subscribe to these events.

Interface ID
`uuid( BECDBE82-3738-4DB2-9BD0-67B3FA7021E5 )`

7.1.2.5.19.1 OnSubscriptionChanged

Signature
```
Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAtscDecAudio_ClipNtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.2.5.19.2 OnBufferAvailable

Signature

```
Void OnBufferAvailable (
    [in] UInt32 cookie,
    [in] UInt32 id );
```

Qualifiers
None.

Description
This function is called by the ATSC Audio Decoder whenever a buffer, which was filled with data, has become free. This may result from decoding of the data filled in this buffer, or from a request from the ATSC Audio Decoder Client to stop the decoding of the audio clip.

The buffers are becoming free for further addition of new data, and are notified, in the same order as they were added by the ATSC Audio Decoder Client.

Parameters

Table 230

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`id`	Identification for the buffer provided by the client.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action

Postcondition
`True`

Remarks
— Each call of `AddData()` with a non-empty buffer eventually results in an `OnBufferAvailable` event.

7.1.2.5.19.3 OnEndOfData

Signature

```
Void OnEndOfData ( [in] UInt32 cookie );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is called by the ATSC audio decoder when it has completed the decoding of a client buffer which has the "`endOfData`" flag set or when it has encountered a client buffer which has the "`endOfData`" flag set with the size of this buffer being 0.

Parameters

Table 231

Name	Description
cookie	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action

Postcondition
`True`

7.1.2.5.19.4 OnStarted

Signature
```
Void OnStarted ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the ATSC Audio decoder if audio clip play has started or continued as a result of the "`Start`" or "`Continue`" function.

Parameters

Table 232

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action

Postcondition
`True`

7.1.2.5.19.5 OnStopped

Signature
```
Void OnStopped ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the ATSC Audio decoder if audio clip play has stopped as a result of the "`Stop`" function.

Parameters

Table 233

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action

Postcondition
`True`

7.1.2.5.19.6 OnPaused

Signature
```
Void OnPaused ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the ATSC Audio decoder if audio clip play has been paused as a result of the "`Pause`" function.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 234

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action

Postcondition
`True`

7.1.3 UhlSpdifDec

7.1.3.1 Concepts

7.1.3.1.1 Introduction

A SPDIF Decoder only consists of a SPDIF audio decoder.

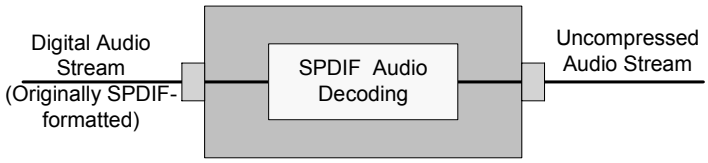


Figure 28 — SPDIF Decoder concepts

7.1.3.1.2 SPDIF Audio Decoding

The function of a SPDIF audio decoder is to decode an originally SPDIF-formatted audio input signal into an uncompressed format suitable for further processing.

The format of the audio input stream is either linear PCM or is encoded audio.

In the first case (linear PCM audio stream), the audio signal is streamed out unchanged.

In the latter case, the audio data complies to the IEC 61937 standard [19] and contains for instance AC-3, MPEG-1, MPEG-2 or DTS encoded audio.

The audio decoder may support one or more audio compression standards. The audio decoder detects into which audio compression standard the digital audio input stream is encoded. If the compression standard is among the compression standards supported by the audio decoder, the audio decoder decodes the digital audio input stream. Clients, when subscribed, receive notifications of changes in the properties of the digital audio input signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The audio decoder has a unique output which is completely determined by the content of its (encoded) audio input stream. This can be a multi-channel, a stereo or a mono audio signal. The number of output channels is identical to the number of channels in the input stream. Channel selection (i.e. downmixing, upmixing) is not provided by the audio decoder.

7.1.3.2 Types & Constants

7.1.3.2.1 Public Types & Constants

7.1.3.2.1.1 Error Codes

None.

7.1.3.2.1.2 uhSpdifDec_ValStat_t

Signature

```
typedef enum _uhSpdifDec_ValStat_t {
    uhSpdifDec_ValStatInvalid = -2,
    uhSpdifDec_ValStatUnknown = -1,
    uhSpdifDec_ValStatKnown   =  1
} uhSpdifDec_ValStat_t, *puhSpdifDec_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 235

Name	Description
uhSpdifDec_ValStatInvalid	The property/properties are temporarily not available.
uhSpdifDec_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhSpdifDec_ValStatKnown	The property/properties are available and determined.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.2.1.3 uhSpdifDec_CompStandard_t

Signature

```
typedef enum _uhSpdifDec_CompStandard_t {
    uhSpdifDec_StandardInvalid = 0x80000000,
    uhSpdifDec_StandardUnknown = 0x40000000,
    uhSpdifDec_Mpeg1Layer1     = 0x00000001,
    uhSpdifDec_Mpeg1Layer2     = 0x00000002,
    uhSpdifDec_Mpeg1Layer3     = 0x00000004,
    uhSpdifDec_Mpeg2Layer1     = 0x00000008,
    uhSpdifDec_Mpeg2Layer2     = 0x00000010,
    uhSpdifDec_Mpeg2Layer3     = 0x00000020,
    uhSpdifDec_Mpeg2Aac        = 0x00000040,
    uhSpdifDec_Ac3             = 0x00000080,
    uhSpdifDec_Dts             = 0x00000100,
    uhSpdifDec_Atrac1          = 0x00000200,
    uhSpdifDec_Atrac2          = 0x00000400,
    uhSpdifDec_Atrac3          = 0x00000800,
    uhSpdifDec_AtracX          = 0x00001000,
    uhSpdifDec_Pcm             = 0x00002000
} uhSpdifDec_CompStandard_t, *puhSpdifDec_CompStandard_t;
```

Qualifiers

— enum-element

Description

Values of this type represent digital audio compression standards.

— The MPEG-1 Standards are defined in [1].

— The MPEG-2 Standards are defined in [2].

— The MPEG-2 AAC Standard is defined in [2].

— The AC-3 Standard is defined in [20].

— The DTS Standard is defined in [7].

Values

Table 236

Name	Description
uhSpdifDec_StandardInvalid	The compression standard property is temporarily not available.
uhSpdifDec_StandardUnknown	The compression standard property is available, yet it could not be determined.
uhSpdifDec_Mpeg1Layer1	MPEG-1, layer I.
uhSpdifDec_Mpeg1Layer2	MPEG-1, layer II.
uhSpdifDec_Mpeg1Layer3	MPEG-1, layer III.
uhSpdifDec_Mpeg2Layer1	MPEG-2, layer I.
uhSpdifDec_Mpeg2Layer2	MPEG-2, layer II.
uhSpdifDec_Mpeg2Layer3	MPEG-2, layer III.
uhSpdifDec_Mpeg2Aac	MPEG-2 AAC.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhSpdifDec_Ac3	AC-3.
uhSpdifDec_Dts	DTS.
uhSpdifDec_Atrac1	ATRAC1.
uhSpdifDec_Atrac2	ATRAC2.
uhSpdifDec_Atrac3	ATRAC3.
uhSpdifDec_AtracX	ATRAC-X.
uhSpdifDec_Pcm	PCM.

7.1.3.2.1.4 uhSpdifDec_CompStandardSet_t

Signature

```
typedef Int32 uhSpdifDec_CompStandardSet_t, *puhSpdifDec_CompStandardSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of digital audio compression standards.

Constraints

None.

7.1.3.2.1.5 uhSpdifDec_SampFreq_t

Signature

```
typedef enum _uhSpdifDec_SampFreq_t {
    uhSpdifDec_FreqInvalid = 0x80000000,
    uhSpdifDec_FreqUnknown = 0x40000000,
    uhSpdifDec_8k          = 0x00000001,
    uhSpdifDec_11k025      = 0x00000002,
    uhSpdifDec_12k         = 0x00000004,
    uhSpdifDec_16k         = 0x00000008,
    uhSpdifDec_22k05       = 0x00000010,
    uhSpdifDec_24k         = 0x00000020,
    uhSpdifDec_32k         = 0x00000040,
    uhSpdifDec_44k1        = 0x00000080,
    uhSpdifDec_48k         = 0x00000100,
    uhSpdifDec_64k         = 0x00000200,
    uhSpdifDec_88k2        = 0x00000400,
    uhSpdifDec_96k         = 0x00000800,
    uhSpdifDec_128k        = 0x00001000,
    uhSpdifDec_176k4       = 0x00002000,
    uhSpdifDec_192k        = 0x00004000
} uhSpdifDec_SampFreq_t, *puhSpdifDec_SampFreq_t;
```

Qualifiers

— enum-element

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sampling frequencies.

Values

Table 237

Name	Description
`uhSpdifDec_FreqInvalid`	The sampling frequency property is temporarily not available.
`uhSpdifDec_FreqUnknown`	The sampling frequency property is available, yet it could not be determined.
`uhSpdifDec_8k`	8 kHz.
`uhSpdifDec_11k025`	11.025 kHz.
`uhSpdifDec_12k`	12 kHz.
`uhSpdifDec_16k`	16 kHz.
`uhSpdifDec_22k05`	22.05 kHz.
`uhSpdifDec_24k`	24 kHz.
`uhSpdifDec_32k`	32 kHz.
`uhSpdifDec_44k1`	44.1 kHz.
`uhSpdifDec_48k`	48 kHz.
`uhSpdifDec_64k`	64 kHz.
`uhSpdifDec_88k2`	88.2 kHz.
`uhSpdifDec_96k`	96 kHz.
`uhSpdifDec_128k`	128 kHz.
`uhSpdifDec_176k4`	176.4 kHz.
`uhSpdifDec_192k`	192 kHz.

7.1.3.2.1.6 uhSpdifDec_SampFreqSet_t

Signature
```
typedef Int32 uhSpdifDec_SampFreqSet_t, *puhSpdifDec_SampFreqSet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of sampling frequencies.

Constraints
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.2.1.7 uhSpdifDec_BitRateInfo_t

Signature

```
typedef enum _uhSpdifDec_BitRateInfo_t {
    uhSpdifDec_VariableInvalid  = -2,
    uhSpdifDec_VariableUnknown  = -1,
    uhSpdifDec_Constant         =  0,
    uhSpdifDec_Variable         =  1
} uhSpdifDec_BitRateInfo_t, *puhSpdifDec_BitRateInfo_t;
```

Qualifiers
None.

Description
Values of this type indicate whether the digital audio input stream has a variable bit rate or not.

Values

Table 238

Name	Description
uhSpdifDec_VariableInvalid	The information over the bit rate variability is temporarily not available.
uhSpdifDec_VariableUnknown	The information over the bit rate variability is available, yet it could not be determined.
uhSpdifDec_Constant	The digital audio input stream has a constant bit rate.
uhSpdifDec_Variable	The digital audio input stream has a variable bit rate.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.2.1.8 uhSpdifDec_ChannelInfo_t

Signature

```
typedef enum _uhSpdifDec_ChannelInfo_t {
    uhSpdifDec_ChannelInvalid      = -2,
    uhSpdifDec_ChannelUnknown      = -1,
    uhSpdifDec_Mono                =  0,
    uhSpdifDec_Stereo              =  1,
    uhSpdifDec_JointStereo         =  2,
    uhSpdifDec_SumDifferenceStereo =  3,
    uhSpdifDec_DolbySurroundStereo =  4,
    uhSpdifDec_2x1                 =  5,
    uhSpdifDec_2x2                 =  6,
    uhSpdifDec_3x0                 =  7,
    uhSpdifDec_3x1                 =  8,
    uhSpdifDec_3x2                 =  9,
    uhSpdifDec_3x3                 = 10,
    uhSpdifDec_3x4                 = 11,
    uhSpdifDec_4x2                 = 12,
    uhSpdifDec_4x4                 = 13,
    uhSpdifDec_5x2                 = 14,
    uhSpdifDec_5x3                 = 15,
    uhSpdifDec_Dual_1x0_1x0        = 16,
    uhSpdifDec_Dual_1x0_2x0        = 17,
    uhSpdifDec_Dual_2x0_2x0        = 18,
    uhSpdifDec_Dual_3x0_2x0        = 19,
    uhSpdifDec_Dual_1x0_1x0_2x0    = 20
} uhSpdifDec_ChannelInfo_t, *puhSpdifDec_ChannelInfo_t;
```

Qualifiers
None.

Description
Values of this type provide information about the channel configuration of the audio input stream.

Values

Table 239

Name	Description
uhSpdifDec_ChannelInvalid	The channel configuration property is temporarily not available.
uhSpdifDec_ChannelUnknown	The channel configuration property is available, yet it could not be determined.
uhSpdifDec_Mono	The audio input stream contains a mono signal.
uhSpdifDec_Stereo	The audio input stream contains a stereo signal. The stereo signal is not Dolby surround encoded or it is not indicated whether Dolby surround is present in the audio input stream.
uhSpdifDec_JointStereo	The audio input stream contains a stereo signal, encoded with a method that exploits stereophonic irrelevance or stereophonic redundancy. This channel configuration can be found for audio signals encoded according to the ATRAC, DTS or the MPEG standard. The decoded audio output stream of the audio decoder contains a plain stereo signal.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhSpdifDec_SumDifferenceStereo	The audio input stream contains a stereo signal, where the left channel contains the sum of the left and right channels of the original audio signal (before encoding), and the right channel the difference between the left and the right channel of the original audio signal (before encoding). This channel configuration can be found for audio signals encoded according to the DTS or the MPEG standard. The decoded audio output stream of the audio decoder contains a plain stereo signal.
uhSpdifDec_DolbySurroundStereo	The audio input stream contains a Dolby surround encoded stereo signal. The output stream of the SPDIF decoder may be further decoded, externally to the SPDIF decoder. If it is decoded by a passive decoder (a.k.a. Dolby Surround decoder), the surround signal is extracted to supplement the two (left and right) front channels; the output stream then contains a front left, a front right, and a rear surround channel. If it is decoded by an active decoder (a.k.a. Dolby Surround Pro Logic decoder and Dolby Pro Logic II decoder), directional enhancement techniques are used to create four separate channels (front left, front center, front right, and surround channel).
uhSpdifDec_2x1	The audio input stream contains a signal with a front left, a front right, and a rear (surround) channel.
uhSpdifDec_2x2	The audio input stream contains a signal with a front left, a front right, a rear (surround) left, and a rear (surround) right channel.
uhSpdifDec_3x0	The audio input stream contains a signal with a front left, a front right, and a front center channel.
uhSpdifDec_3x1	The audio input stream contains a signal with a front left, a front right, a front center, and a rear (surround) channel.
uhSpdifDec_3x2	The audio input stream contains a signal with a front left, a front right, a front center, a rear (surround) left, and a rear (surround) right channel.
uhSpdifDec_3x3	The audio input stream contains a signal with a front left, a front right, a front center, a rear (surround) left, a rear (surround) right, and a rear (surround) center channel.
uhSpdifDec_3x4	The audio input stream contains a signal with a front left, a front right, a front center, a left surround, a left-back surround, a right-back surround, and a right surround channel.
uhSpdifDec_4x2	The audio input stream contains a signal with a front left, a front left-center, a front right-center, a front right, a rear (surround) left, and a rear (surround) right channel.
uhSpdifDec_4x4	The audio input stream contains a signal with a front left, a front left-center, a front right-center, a front right, a first rear (surround) left, a second rear (surround) left, a first rear (surround) right, and a second rear (surround) right channel.
uhSpdifDec_5x2	The audio input stream contains a signal with a front left, a front left-center, a center, a front right-center, a front right, a rear (surround) left, and a rear (surround) right channel.
uhSpdifDec_5x3	The audio input stream contains a signal with a front left, a front left-center, a center, a front right-center, a front right, a rear (surround) left, a rear (surround) right, and a rear (surround) center channel.
uhSpdifDec_Dual_1x0_1x0	The audio input stream contains two mono signals with independent content.
uhSpdifDec_Dual_1x0_2x0	The audio input stream contains a mono and a stereo signal with independent content. This channel configuration can be found for audio signals encoded according to the MPEG-2 standard.
uhSpdifDec_Dual_2x0_2x0	The audio input stream contains two stereo signals with independent content. This channel configuration can be found for audio signals encoded according to the MPEG-2 standard.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhSpdifDec_Dual_3x0_2x0	The audio input stream contains a signal with a front left, a front right, and a front center channel and a stereo signal. The two signals have independent contents. This channel configuration can be found for audio signals encoded according to the MPEG-2 standard.
uhSpdifDec_Dual_1x0_1x0_2x0	The audio input stream contains two mono signals and a stereo signal. The three signals have independent contents. This channel configuration can be found for audio signals encoded according to the MPEG-2 standard.

7.1.3.2.1.9 uhSpdifDec_LfeInfo_t

Signature

```
typedef enum _uhSpdifDec_LfeInfo_t {
    uhSpdifDec_LfePresentInvalid = -2,
    uhSpdifDec_LfePresentUnknown = -1,
    uhSpdifDec_LfeNotPresent     =  0,
    uhSpdifDec_LfePresent        =  1
} uhSpdifDec_LfeInfo_t, *puhSpdifDec_LfeInfo_t;
```

Qualifiers
None.

Description
Values of this type indicate whether a LFE channel is present in the digital audio input stream or not.

Values

Table 240

Name	Description
uhSpdifDec_LfePresentInvalid	The information over the presence of the LFE channel is temporarily not available.
uhSpdifDec_LfePresentUnknown	The information over the presence of the LFE channel is available, yet it could not be determined.
uhSpdifDec_LfeNotPresent	The digital audio input stream does not contain a LFE channel.
uhSpdifDec_LfePresent	The digital audio input stream contains a LFE channel.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.2.1.10 uhSpdifDec_Language_t

Signature

```
typedef enum _uhSpdifDec_Language_t {
    uhSpdifDec_LanguageInvalid =  -2,
    uhSpdifDec_LanguageUnknown =  -1,
    uhSpdifDec_Albanian        =   1,
    uhSpdifDec_Breton          =   2,
    uhSpdifDec_Catalan         =   3,
    uhSpdifDec_Croatian        =   4,
    uhSpdifDec_Welsh           =   5,
    uhSpdifDec_Czech           =   6,
    uhSpdifDec_Danish          =   7,
    uhSpdifDec_German          =   8,
    uhSpdifDec_English         =   9,
    uhSpdifDec_Spanish         =  10,
    uhSpdifDec_Esperanto       =  11,
    uhSpdifDec_Estonian        =  12,
    uhSpdifDec_Basque          =  13,
    uhSpdifDec_Faroese         =  14,
    uhSpdifDec_French          =  15,
    uhSpdifDec_Frisian         =  16,
    uhSpdifDec_Irish           =  17,
    uhSpdifDec_Gaelic          =  18,
    uhSpdifDec_Galician        =  19,
    uhSpdifDec_Icelandic       =  20,
    uhSpdifDec_Italian         =  21,
    uhSpdifDec_Lappish         =  22,
    uhSpdifDec_Latin           =  23,
    uhSpdifDec_Latvian         =  24,
    uhSpdifDec_Luxembourgian   =  25,
    uhSpdifDec_Lithuanian      =  26,
    uhSpdifDec_Hungarian       =  27,
    uhSpdifDec_Maltese         =  28,
    uhSpdifDec_Dutch           =  29,
    uhSpdifDec_Norwegian       =  30,
    uhSpdifDec_Occitan         =  31,
    uhSpdifDec_Polish          =  32,
    uhSpdifDec_Portuguese      =  33,
    uhSpdifDec_Romanian        =  34,
    uhSpdifDec_Romansh         =  35,
    uhSpdifDec_Serbian         =  36,
    uhSpdifDec_Slovak          =  37,
    uhSpdifDec_Slovene         =  38,
    uhSpdifDec_Finnish         =  39,
    uhSpdifDec_Swedish         =  40,
    uhSpdifDec_Turkish         =  41,
    uhSpdifDec_Flemish         =  42,
    uhSpdifDec_Walloon         =  43,
    uhSpdifDec_BackgroundSound =  64,
    uhSpdifDec_Zulu            =  69,
    uhSpdifDec_Vietnamese      =  70,
    uhSpdifDec_Uzbek           =  71,
    uhSpdifDec_Urdu            =  72,
    uhSpdifDec_Ukrainian       =  73,
    uhSpdifDec_Thai            =  74,
    uhSpdifDec_Telugu          =  75,
    uhSpdifDec_Tatar           =  76,
    uhSpdifDec_Tamil           =  77,
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    uhSpdifDec_Tadzhik        =  78,
    uhSpdifDec_Swahili        =  79,
    uhSpdifDec_SrananTongo    =  80,
    uhSpdifDec_Somali         =  81,
    uhSpdifDec_Sinhalese      =  82,
    uhSpdifDec_Shona          =  83,
    uhSpdifDec_SerboCroat     =  84,
    uhSpdifDec_Ruthenian      =  85,
    uhSpdifDec_Russian        =  86,
    uhSpdifDec_Quechua        =  87,
    uhSpdifDec_Pustu          =  88,
    uhSpdifDec_Punjabi        =  89,
    uhSpdifDec_Persian        =  90,
    uhSpdifDec_Papiamento     =  91,
    uhSpdifDec_Oriya          =  92,
    uhSpdifDec_Nepali         =  93,
    uhSpdifDec_Ndebele        =  94,
    uhSpdifDec_Marathi        =  95,
    uhSpdifDec_Moldavian      =  96,
    uhSpdifDec_Malaysian      =  97,
    uhSpdifDec_Malagasay      =  98,
    uhSpdifDec_Macedonian     =  99,
    uhSpdifDec_Laotian        = 100,
    uhSpdifDec_Korean         = 101,
    uhSpdifDec_Khmer          = 102,
    uhSpdifDec_Kazakh         = 103,
    uhSpdifDec_Kannada        = 104,
    uhSpdifDec_Japanese       = 105,
    uhSpdifDec_Indonesian     = 106,
    uhSpdifDec_Hindi          = 107,
    uhSpdifDec_Hebrew         = 108,
    uhSpdifDec_Hausa          = 109,
    uhSpdifDec_Gurani         = 110,
    uhSpdifDec_Gujurati       = 111,
    uhSpdifDec_Greek          = 112,
    uhSpdifDec_Georgian       = 113,
    uhSpdifDec_Fulani         = 114,
    uhSpdifDec_Dari           = 115,
    uhSpdifDec_Churash        = 116,
    uhSpdifDec_Chinese        = 117,
    uhSpdifDec_Burmese        = 118,
    uhSpdifDec_Bulgarian      = 119,
    uhSpdifDec_Bengali        = 120,
    uhSpdifDec_Byelorussian   = 121,
    uhSpdifDec_Bambora        = 122,
    uhSpdifDec_Azerbaijani    = 123,
    uhSpdifDec_Assamese       = 124,
    uhSpdifDec_Armenian       = 125,
    uhSpdifDec_Arabic         = 126,
    uhSpdifDec_Amharic        = 127,
    uhSpdifDec_OtherLanguage  = 128
} uhSpdifDec_Language_t, *puhSpdifDec_Language_t;
```

Qualifiers
None.

Description
Values of this type represent languages.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 241

Name	Description
uhSpdifDec_LanguageInvalid	The language property is temporarily not available.
uhSpdifDec_LanguageUnknown	The language property is available, yet it could not be determined.
uhSpdifDec_Albanian	Albanian language.
uhSpdifDec_Breton	Breton language.
uhSpdifDec_Catalan	Catalan language.
uhSpdifDec_Croatian	Croatian language.
uhSpdifDec_Welsh	Welsh language.
uhSpdifDec_Czech	Czech language.
uhSpdifDec_Danish	Danish language.
uhSpdifDec_German	German language.
uhSpdifDec_English	English language.
uhSpdifDec_Spanish	Spanish language.
uhSpdifDec_Esperanto	Esperanto language.
uhSpdifDec_Estonian	Estonian language.
uhSpdifDec_Basque	Basque language.
uhSpdifDec_Faroese	Faroese language.
uhSpdifDec_French	French language.
uhSpdifDec_Frisian	Frisian language.
uhSpdifDec_Irish	Irish language.
uhSpdifDec_Gaelic	Gaelic language.
uhSpdifDec_Galician	Galician language.
uhSpdifDec_Icelandic	Icelandic language.
uhSpdifDec_Italian	Italian language.
uhSpdifDec_Lappish	Lappish language.
uhSpdifDec_Latin	Latin language.
uhSpdifDec_Latvian	Latvian language.
uhSpdifDec_Luxembourgian	Luxembourgian language.
uhSpdifDec_Lithuanian	Lithuanian language.
uhSpdifDec_Hungarian	Hungarian language.
uhSpdifDec_Maltese	Maltese language.
uhSpdifDec_Dutch	Dutch language.
uhSpdifDec_Norwegian	Norwegian language.
uhSpdifDec_Occitan	Occitan language.
uhSpdifDec_Polish	Polish language.
uhSpdifDec_Portuguese	Portuguese language.
uhSpdifDec_Romanian	Romanian language.
uhSpdifDec_Romansh	Romansh language.
uhSpdifDec_Serbian	Serbian language.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhSpdifDec_Slovak	Slovak language.
uhSpdifDec_Slovene	Slovene language.
uhSpdifDec_Finnish	Finnish language.
uhSpdifDec_Swedish	Swedish language.
uhSpdifDec_Turkish	Turkish language.
uhSpdifDec_Flemish	Flemish language.
uhSpdifDec_Walloon	Walloon language.
uhSpdifDec_BackgroundSound	Background sound/clean feed.
uhSpdifDec_Zulu	Zulu language.
uhSpdifDec_Vietnamese	Vietnamese language.
uhSpdifDec_Uzbek	Uzbek language.
uhSpdifDec_Urdu	Urdu language.
uhSpdifDec_Ukrainian	Ukrainian language.
uhSpdifDec_Thai	Thai language.
uhSpdifDec_Telugu	Telugu language.
uhSpdifDec_Tatar	Tatar language.
uhSpdifDec_Tamil	Tamil language.
uhSpdifDec_Tadzhik	Tadzhik language.
uhSpdifDec_Swahili	Swahili language.
uhSpdifDec_SrananTongo	Sranan Tongo language.
uhSpdifDec_Somali	Somali language.
uhSpdifDec_Sinhalese	Sinhalese language.
uhSpdifDec_Shona	Shona language.
uhSpdifDec_SerboCroat	Serbo-Croat language.
uhSpdifDec_Ruthenian	Ruthenian language.
uhSpdifDec_Russian	Russian language.
uhSpdifDec_Quechua	Quechua language.
uhSpdifDec_Pustu	Pustu language.
uhSpdifDec_Punjabi	Punjabi language.
uhSpdifDec_Persian	Persian language.
uhSpdifDec_Papiamento	Papiamento language.
uhSpdifDec_Oriya	Oriya language.
uhSpdifDec_Nepali	Nepali language.
uhSpdifDec_Ndebele	Ndebele language.
uhSpdifDec_Marathi	Marathi language.
uhSpdifDec_Moldavian	Moldavian language.
uhSpdifDec_Malaysian	Malaysian language.
uhSpdifDec_Malagasay	Malagasay language.
uhSpdifDec_Macedonian	Macedonian language.
uhSpdifDec_Laotian	Laotian language.
uhSpdifDec_Korean	Korean language.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhSpdifDec_Khmer	Khmer language.
uhSpdifDec_Kazakh	Kazakh language.
uhSpdifDec_Kannada	Kannada language.
uhSpdifDec_Japanese	Japanese language.
uhSpdifDec_Indonesian	Indonesian language.
uhSpdifDec_Hindi	Hindi language.
uhSpdifDec_Hebrew	Hebrew language.
uhSpdifDec_Hausa	Hausa language.
uhSpdifDec_Gurani	Gurani language.
uhSpdifDec_Gujurati	Gujurati language.
uhSpdifDec_Greek	Greek language.
uhSpdifDec_Georgian	Georgian language.
uhSpdifDec_Fulani	Fulani language.
uhSpdifDec_Dari	Dari language.
uhSpdifDec_Churash	Churash language.
uhSpdifDec_Chinese	Chinese language.
uhSpdifDec_Burmese	Burmese language.
uhSpdifDec_Bulgarian	Bulgarian language.
uhSpdifDec_Bengali	Bengali language.
uhSpdifDec_Byelorussian	Byelorussian language.
uhSpdifDec_Bambora	Bambora language.
uhSpdifDec_Azerbaijani	Azerbaijani language.
uhSpdifDec_Assamese	Assamese language.
uhSpdifDec_Armenian	Armenian language.
uhSpdifDec_Arabic	Arabic language.
uhSpdifDec_Amharic	Amharic language.
uhSpdifDec_OtherLanguage	Another language than the ones explicitly defined in the type.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.2.1.11 uhSpdifDec_ServiceType_t

Signature

```
typedef enum _uhSpdifDec_ServiceType_t {
    uhSpdifDec_ServiceInvalid  = -2,
    uhSpdifDec_ServiceUnknown  = -1,
    uhSpdifDec_CompleteMain    =  0,
    uhSpdifDec_MusicAndEffects =  1,
    uhSpdifDec_VisualImpaired  =  2,
    uhSpdifDec_HearingImpaired =  3,
    uhSpdifDec_CleanEffects    =  4,
    uhSpdifDec_Dialogue        =  5,
    uhSpdifDec_Commentary      =  6,
    uhSpdifDec_Emergency       =  7,
    uhSpdifDec_VoiceOver       =  8,
    uhSpdifDec_Karaoke         =  9
} uhSpdifDec_ServiceType_t, *puhSpdifDec_ServiceType_t;
```

Qualifiers
None.

Description
Values of this type provide information about the type of audio service that the digital audio input stream conveys.

Values

Table 242

Name	Description
uhSpdifDec_ServiceInvalid	The audio service property is temporarily not available.
uhSpdifDec_ServiceUnknown	The audio service property is available, yet it could not be determined.
uhSpdifDec_CompleteMain	Complete Main.
uhSpdifDec_MusicAndEffects	Music and Effects.
uhSpdifDec_VisualImpaired	The audio content is prepared for the visually impaired viewer.
uhSpdifDec_HearingImpaired	The audio content is prepared for hearing impaired.
uhSpdifDec_CleanEffects	The audio content has no language.
uhSpdifDec_Dialogue	Dialogue.
uhSpdifDec_Commentary	Commentary.
uhSpdifDec_Emergency	Emergency.
uhSpdifDec_VoiceOver	Voice-Over.
uhSpdifDec_Karaoke	Karaoke.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.2.1.12 uhSpdifDec_Error_t

Signature

```
typedef enum _uhSpdifDec_Error_t {
    uhSpdifDec_NoError       = 0x00000000,
    uhSpdifDec_SyncNotFound  = 0x00000001,
    uhSpdifDec_InvalidHeader = 0x00000002,
    uhSpdifDec_DecodeFrame   = 0x00000004,
    uhSpdifDec_CrcFailed     = 0x00000008,
    uhSpdifDec_Underrun      = 0x00000010
} uhSpdifDec_Error_t, *puhSpdifDec_Error_t;
```

Qualifiers

— enum-element

Description

This type enumerates the errors that can occur in the SPDIF decoder.

Values

Table 243

Name	Description
uhSpdifDec_NoError	Indicates that there is no error.
uhSpdifDec_SyncNotFound	The audio decoder did not find a valid sync word within the search range. To maintain a continuous data stream at the output the audio decoder may mute one frame.
uhSpdifDec_InvalidHeader	Invalid header. Error while parsing the bitstream.
uhSpdifDec_DecodeFrame	An error occurred while decoding. The audio decoder may perform muting of the current audio block, and the rest of the frame.
uhSpdifDec_CrcFailed	The calculation of the CRC derived a wrong value. Audio block muting or audio block repeating may be performed.
uhSpdifDec_Underrun	The audio decoder requested data, but none was available.

7.1.3.2.1.13 uhSpdifDec_ErrorSet_t

Signature

```
typedef Int32 uhSpdifDec_ErrorSet_t, *puhSpdifDec_ErrorSet_t;
```

Qualifiers

— enum-set

Description

This type represents sets of errors that can occur in the SPDIF decoder. Values of this type are typically constructed by logical OR-ing of values of type uhSpdifDec_Error_t.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.2.1.14 uhSpdifDec_Ntf_t

Signature

```
typedef enum _uhSpdifDec_Ntf_t {
    uhSpdifDec_SubscriptionChanged  = 0x00000001,
    uhSpdifDec_StandardChanged      = 0x00000002,
    uhSpdifDec_SampFreqChanged      = 0x00000004,
    uhSpdifDec_BitRateChanged       = 0x00000008,
    uhSpdifDec_ChannelsChanged      = 0x00000010,
    uhSpdifDec_LanguageChanged      = 0x00000020,
    uhSpdifDec_ServiceTypeChanged   = 0x00000040,
    uhSpdifDec_Error                = 0x00000080,
    uhSpdifDec_Recovered            = 0x00000100,
    uhSpdifDec_StableOutput         = 0x00000200
} uhSpdifDec_Ntf_t, *puhSpdifDec_Ntf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhISpdifDecNtf`.

Values

Table 244

Name	Description
`uhSpdifDec_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhSpdifDec_StandardChanged`	Represents `OnStandardChanged`.
`uhSpdifDec_SampFreqChanged`	Represents `OnSampFreqChanged`.
`uhSpdifDec_BitRateChanged`	Represents `OnBitRateChanged`.
`uhSpdifDec_ChannelsChanged`	Represents `OnChannelsChanged`.
`uhSpdifDec_LanguageChanged`	Represents `OnLanguageChanged`.
`uhSpdifDec_ServiceTypeChanged`	Represents `OnServiceTypeChanged`.
`uhSpdifDec_Error`	Represents `OnError`. Some error(s) occurred while decoding the incoming signal. The exact error(s) that occurred will be an argument of the `OnError` notification that is generated (see type uhSpdifDec_Error_t).
`uhSpdifDec_Recovered`	Represents `OnRecovered`.
`uhSpdifDec_StableOutput`	Represents `OnStableOutput`.

7.1.3.2.1.15 uhSpdifDec_NtfSet_t

Signature

```
typedef Int32 uhSpdifDec_NtfSet_t, *puhSpdifDec_NtfSet_t;
```

Qualifiers

— enum-set

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhSpdifDec_Ntf_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

Constraints
None.

7.1.3.2.1.16 uhSpdifDec_AllNtfs

Signature
```
const uhSpdifDec_NtfSet_t uhSpdifDec_AllNtfs = 0x000003FF;
```

Qualifiers
None.

Description
Defines the set of all values of type uhSpdifDec_Ntf_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

7.1.3.2.2 Model Types & Constants

None.

7.1.3.3 Logical Component

7.1.3.3.1 Interface-Role Model

Figure 29 depicts the interface-role model of the SPDIF Decoder logical component (the grey box in the interface-role model). It shows the interfaces and the roles involved with this component.

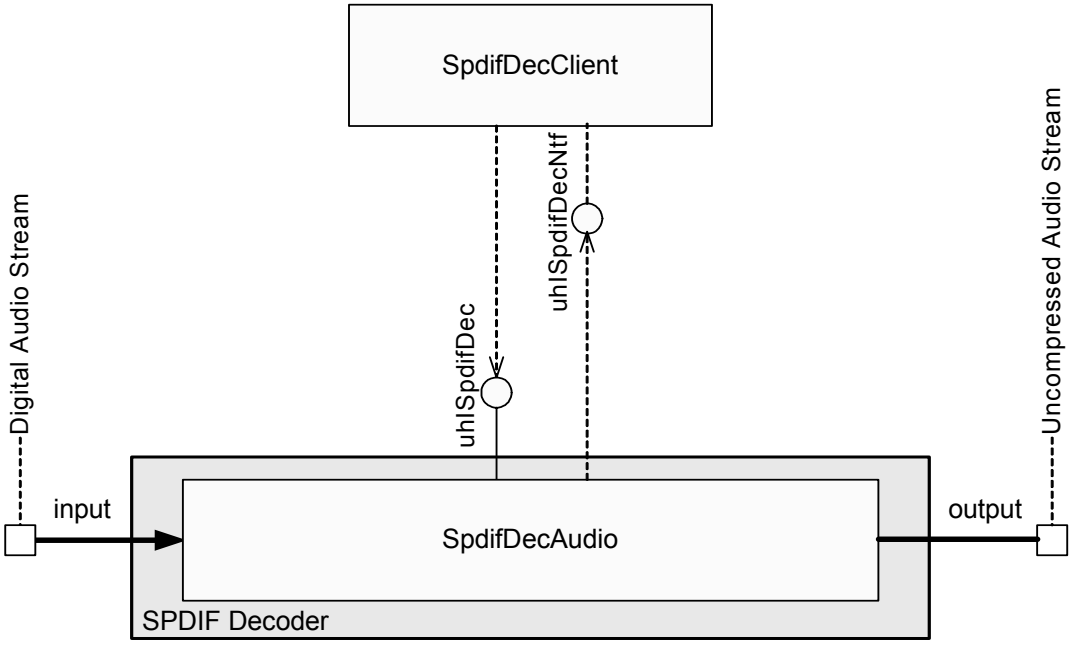


Figure 29 — Interface Role Model of a SPDIF Decoder

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The SPDIF Decoder logical component contains a single `SpdifDecAudio` role. This role provides one mandatory (solid line) control interface to its client. A client can optionally provide a call-back interface allowing it to receive notifications of events in the `SpdifDecAudio`.

7.1.3.3.2 Diversity

7.1.3.3.2.1 Provided Interfaces

Table 245

Role	Interface	Presence
`SpdifDecAudio`	`uhISpdifDec`	mandatory
`SpdifDecClient`	`uhISpdifDecNtf`	optional

7.1.3.3.2.2 Configurable Items

Table 246

Role	Attribute
`SpdifDecAudio`	`suppCompStandards`
	`suppSampFreqs`

7.1.3.3.2.3 Constraints

```
—— !( suppCompStandards & uhSpdifDec_StandardInvalid)
—— !( suppCompStandards & uhSpdifDec_StandardUnknown)
—— For each s in uhSpdifDec_CompStandard_t
   —— If s in suppCompStandards
      —— !( suppSampFreqs[s] & uhSpdifDec_FreqInvalid)
      —— !( suppSampFreqs[s] & uhSpdifDec_FreqUnknown)
   —— else
      —— suppSampFreqs[s] == uhSpdifDec_FreqUnknown
```

7.1.3.3.3 Instantiation

7.1.3.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 247

Type	Object	Multiplicity
`SpdifDecAudio`	`dec`	1

7.1.3.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
—— dec.compStandard == uhSpdifDec_StandardInvalid
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— dec.sampFreq == uhSpdifDec_FreqInvalid
— dec.variableBitRate == uhSpdifDec_VariableInvalid
— dec.bitRateStatus == uhSpdifDec_ValStatInvalid
— dec.bitRate == 0
— dec.channelInfo == uhSpdifDec_ChannelInvalid
— dec.lfePresent == uhSpdifDec_LfePresentInvalid
— dec.primaryLanguage == uhSpdifDec_LanguageInvalid
— dec.secondaryLanguage == uhSpdifDec_LanguageInvalid
— dec.serviceType == uhSpdifDec_ServiceInvalid
— dec.errorStatus == uhSpdifDec_NoError
— dec.stableOutput == False

7.1.3.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

7.1.3.4 Roles

7.1.3.4.1 SpdifDecAudio

Signature

```
role SpdifDecAudio {
    const uhSpdifDec_CompStandardSet_t   suppCompStandards;
    uhSpdifDec_CompStandard_t            compStandard;
    const                                        uhSpdifDec_SampFreqSet_t
suppSampFreqs[uhSpdifDec_CompStandard_t];
    uhSpdifDec_SampFreq_t                sampFreq;
    uhSpdifDec_BitRateInfo_t             variableBitRate;
    uhSpdifDec_ValStat_t                 bitRateStatus;
    UInt32                               bitRate;
    uhSpdifDec_ChannelInfo_t             channelInfo;
    uhSpdifDec_LfeInfo_t                 lfePresent;
    uhSpdifDec_Language_t                primaryLanguage;
    uhSpdifDec_Language_t                secondaryLanguage;
    uhSpdifDec_ServiceType_t             serviceType;
    uhSpdifDec_ErrorSet_t                errorStatus;
    Bool                                 stableOutput;
}
```

Qualifiers

— root

Description

A SPDIF audio decoder decodes an originally SPDIF-formatted encoded audio input stream and outputs a decoded audio stream. It autonomously detects the compression standard of the encoded audio input stream and provides notifications of various changes that occur in the input stream.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 248

Name	Description
suppCompStandards	The set of compression standards that the audio decoder supports.
compStandard	The audio compression standard of the digital audio input stream.
suppSampFreqs	For each compression standard, the set of sampling frequencies that the audio decoder supports.
sampFreq	The sampling frequency of the digital audio input stream.
variableBitRate	Indicates whether the digital audio input stream has a variable bit rate or not.
bitRateStatus	Validity state of the bit rate in birRate.
bitRate	The bit rate (in kbit/s) of the digital audio input stream.
channelInfo	The channel configuration of the digital audio input stream.
lfePresent	Indicates whether a LFE channel is present in the digital audio input stream or not.
primaryLanguage	The primary language of the digital audio input stream.
secondaryLanguage	The secondary language of the digital audio input stream.
serviceType	The type of audio service that the digital audio input stream conveys.
errorStatus	The current error status, represented by the set of errors that have occurred, and from which the SPDIF decoder has not yet recovered.
stableOutput	Indicates whether the SPDIF decoder is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream (True if this is the case, False otherwise).

Dependent Attributes

None.

Invariants

```
— If ((variableBitRate == uhSpdifDec_VariableUnknown) || (variableBitRate ==
  uhSpdifDec_Variable))
  — bitRateStatus == uhSpdifDec_ValStatUnknown
— If ((channelInfo != uhSpdifDec_Dual_1x0_1x0) &&
  (channelInfo != uhSpdifDec_Dual_1x0_2x0) &&
  (channelInfo != uhSpdifDec_Dual_2x0_2x0) &&
  (channelInfo != uhSpdifDec_Dual_3x0_2x0) &&
  (channelInfo != uhSpdifDec_Dual_1x0_1x0_2x0))
  — secondaryLanguage == uhSpdifDec_LanguageUnknown
— If (compStandard == uhSpdifDec_StandardInvalid)
  — sampFreq == uhSpdifDec_FreqInvalid
— If (sampFreq == uhSpdifDec_FreqInvalid)
  — variableBitRate == uhSpdifDec_VariableInvalid
  — bitRateStatus == uhSpdifDec_ValStatInvalid
  — channelInfo == uhSpdifDec_ChannelInvalid
  — lfePresent == uhSpdifDec_LfePresentInvalid
  — primaryLanguage == uhSpdifDec_LanguageInvalid
  — secondaryLanguage == uhSpdifDec_LanguageInvalid
  — serviceType == uhSpdifDec_ServiceInvalid
— If (compStandard == uhSpdifDec_StandardUnknown)
  — sampFreq == uhSpdifDec_FreqUnknown
— If (sampFreq == uhSpdifDec_FreqUnknown)
  — variableBitRate == uhSpdifDec_VariableUnknown
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- bitRateStatus == uhSpdifDec_ValStatUnknown
- channelInfo == uhSpdifDec_ChannelUnknown
- lfePresent == uhSpdifDec_LfePresentUnknown
- primaryLanguage == uhSpdifDec_LanguageUnknown
- secondaryLanguage == uhSpdifDec_LanguageUnknown
- serviceType == uhSpdifDec_ServiceUnknown

Instantiation
SpdifDecAudio is always instantiated as part of a SPDIF Decoder; see subclause 3.3.

Streaming Behavior

- The audio decoder continually observes the input stream.

- When an input stream is detected the audio decoder will try to detect the compression standard of the input stream and set the value compStandard accordingly. As long as no compression standard has been detected the value of compStandard is equal to uhSpdifDec_StandardInvalid. If the compression standard is detected but is not in the set suppCompStandards, then the value of compStandard is set to uhSpdifDec_StandardUnknown. When a supported compression standard *cs* is detected the value of compStandard is set to *cs*.

- If the sampling frequency of the audio input stream is not in the set suppSampFreqs[cs], the value of sampFreq is set to uhSpdifDec_FreqUnknown and the audio input stream is not decoded. As soon as the sampling frequency *sf* of the input stream is in the set suppSampFreqs[cs], the value of sampFreq is set to *sf* and the input stream is decoded.

- When the audio input signal drops or when the compression standard changes and is not yet detected, the value of compStandard is set to uhSpdifDec_StandardInvalid, the values of the other attributes of the audio decoder are also set to Invalid (including sampFreq, which is set to uhSpdifDec_FreqInvalid). Detection of the compression standard of the audio input stream is restarted; no output data is then produced.

After that, when the compression standard of the digital audio input stream can't be determined, or is not in the set suppCompStandards, the value of compStandard is then reset to uhSpdifDec_StandardUnknown. The values of the other attributes of the audio decoder are then also reset to Unknown (including sampFreq which is then reset to uhSpdifDec_FreqUnknown). Once again, no output data is produced.

Output data is also not produced when the sampling frequency of the audio input stream is not in the set suppSampFreqs[cs] although the compression standard *cs* of the audio input stream could be determined and could be in the set suppCompStandards. The value of sampFreq is then set to uhSpdifDec_FreqUnknown. The values of the other attributes of the audio decoder (excluding compStandard) are then also reset to Unknown.

- In all cases, the audio decoder keeps reading and parsing the incoming audio packets, looks for a valid compression standard and for a valid sampling frequency.

- When one or more changes occur in the properties of the audio input stream, the audio decoder provides a notification to the subscribed client(s). If the changes in the properties may result in artefacts such as e.g. blops, ticks if the audio decoder continues producing output data, then the audio decoder stops producing output data, a soft mute (a cosine mute, for instance) is applied, the audio decoder determines when it is possible to produce output data again, and a soft unmute (a cosine unmute, for instance) is accordingly applied.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Active Behavior
The audio decoder has standard notification behavior (see `uhISpdifDecNtf`), apart from the following events.

Table 249

Event	Action
The SPDIF decoder encounters one or more errors that are not part of `dec.errorStatus`.	— Let *e* be the set of the one or more errors that the SPDIF decoder encountered and that are not part of `dec.errorStatus`. — Add the errors in *e* to `dec.errorStatus` — notify `OnError(*, e)`
The SPDIF decoder recovers from one or more errors. NOTE It is up to the implementation to decide when it has recovered from an error.	— Let *e* be the set of the one or more errors that the SPDIF decoder recovered from. — Remove the errors in *e* from `dec.errorStatus` — notify `OnRecovered(*, e)`
The SPDIF decoder is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream. (`dec.stableOutput == False` when this event occurs).	`dec.stableOutput = True` — notify `OnStableOutput(*, True)`
The SPDIF decoder is no longer outputting a stable (decoded) audio stream (`dec.stableOutput == True` when this event occurs).	`dec.stableOutput = False` — notify `OnStableOutput(*, False)`

Execution Constraints
None.

7.1.3.4.2 SpdifDecClient

Signature
```
role SpdifDecClient {}
```

Qualifiers
— actor

Description
A SPDIF decoder client represents a user of the `uhISpdifDec` interface.

Execution Constraints
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.5 Interfaces

7.1.3.5.1 uhISpdifDec

Qualifiers
— None.

Description
This interface of a `SpdifDecAudio` provides functions to retrieve properties of the digital audio input stream and provides a subscription and an unsubscription function on notifications of changes of these properties.

Interface ID
`uuid( FB46ED44-AED1-497d-9967-3FDE251D5E11 )`

7.1.3.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhISpdifDecNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhSpdifDec_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.1.3.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhISpdifDecNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhSpdifDec_NtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

7.1.3.5.1.3 GetSuppStandards

Signature
```
uhErrorCode_t GetSuppStandards (
    [out] puhSpdifDec_CompStandardSet_t pStandards );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the set of compression standards supported by the audio decoder.

Parameters

Table 250

Name	Description
`pStandards`	Output parameter returning the set of compression standards that the audio decoder supports.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*`pStandards `== dec.suppCompStandards`

7.1.3.5.1.4 GetStandard

Signature
```
uhErrorCode_t GetStandard ( [out] puhSpdifDec_CompStandard_t pStandard );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the compression standard of the digital audio input stream.

Parameters

Table 251

Name	Description
`pStandard`	Output parameter returning the compression standard of the digital audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `*pStandard == dec.compStandard`

7.1.3.5.1.5 GetSuppSampFreqs

Signature
```
uhErrorCode_t GetSuppSampFreqs (
    [in] uhSpdifDec_CompStandard_t standard,
    [out] puhSpdifDec_SampFreqSet_t pSampFreqs );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the set of sampling frequencies supported by the audio decoder for the specified compression standard.

Parameters

Table 252

Name	Description
`standard`	The compression standard for which the set of supported frequencies is requested.
`pSampFreqs`	Output parameter returning the set of sampling frequencies that the audio decoder supports for the compression standard `standard`.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pSampFreqs == dec.suppSampFreqs[standard]`

7.1.3.5.1.6 GetSampFreq

Signature
```
uhErrorCode_t GetSampFreq ( [out] puhSpdifDec_SampFreq_t pSampFreq );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Returns the sampling frequency of the digital audio input stream.

Parameters

Table 253

Name	Description
`pSampFreq`	Output parameter returning the sampling frequency of the digital audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— *`*pSampFreq == dec.sampFreq`*

7.1.3.5.1.7 GetBitRate

Signature
```
uhErrorCode_t GetBitRate (
    [out] puhSpdifDec_BitRateInfo_t pVariable,
    [out] puhSpdifDec_ValStat_t     pStatus,
    [out] pUInt32                   pBitRate );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the bit rate of the digital audio input stream is variable or constant. Also returns the bit rate (in kbit/s) of the digital audio input stream.

Parameters

Table 254

Name	Description
`pVariable`	Output parameter returning whether the bit rate of the digital audio input stream is a variable bit rate (VBR) or not.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
pStatus	Output parameter that indicates the validity of the bit rate parameter pBitRate.
pBitRate	Output parameter returning the bit rate (in kbit/s) of the digital audio input stream.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pVariable == dec.variableBitRate
— *pStatus == dec.bitRateStatus
— **pBitRate* == dec.bitRate

7.1.3.5.1.8 GetChannels

Signature
```
uhErrorCode_t GetChannels (
    [out] puhSpdifDec_ChannelInfo_t pChannelInfo,
    [out] puhSpdifDec_LfeInfo_t pLfePresent );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the channel configuration of the digital audio input stream. Also returns whether a LFE channel is present in the digital audio input stream.

Parameters

Table 255

Name	Description
pChannelInfo	Output parameter returning the channel configuration of the digital audio input stream.
pLfePresent	Output parameter returning whether a LFE channel is present in the digital audio input stream.

Return Values
Standard.

Precondition
True

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— `*pChannelInfo == dec.channelInfo`
— `*pLfePresent == dec.lfePresent`

7.1.3.5.1.9 GetLanguage

Signature
```
uhErrorCode_t GetLanguage (
    [out] puhSpdifDec_Language_t pPrimaryLanguage,
    [out] puhSpdifDec_Language_t pSecondaryLanguage );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the primary and secondary languages of the digital audio input stream.

Parameters

Table 256

Name	Description
`pPrimaryLanguage`	Output parameter returning the primary language of the digital audio input stream.
`pSecondaryLanguage`	Output parameter returning the secondary language of the digital audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pPrimaryLanguage == dec.primaryLanguage`
— `*pSecondaryLanguage == dec.secondaryLanguage`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.3.5.1.10 GetServiceType

Signature
```
uhErrorCode_t GetServiceType ( [out] puhSpdifDec_ServiceType_t pServiceType );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the type of the audio service that the digital audio input stream conveys.

Parameters

Table 257

Name	Description
`pServiceType`	Output parameter returning the type of the audio service that the digital audio input stream conveys.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— *pServiceType == `dec.serviceType`

7.1.3.5.1.11 GetErrorStatus

Signature
```
uhErrorCode_t GetErrorStatus (
    [out] puhSpdifDec_ErrorSet_t pErrorStatus );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current error status.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 258

Name	Description
`pErrorStatus`	Output parameter returning the current error status.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pErrorStatus == dec.errorStatus`

7.1.3.5.1.12 GetStableOutput

Signature
```
uhErrorCode_t GetStableOutput (
    [out] pBool pStableOutput);
```

Qualifiers
- synchronous
- thread-safe

Description
Returns whether the SPDIF decoder has currently stable output.

Parameters

Table 259

Name	Description
`pStableOutput`	Output parameter returning whether the SPDIF decoder has currently stable output.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
```
*pStableOutput == dec.stableOutput
```

7.1.3.5.2 uhISpdifDecNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in the `SpdifDecAudio`. The `Subscribe` function of `uhISpdifDec` interface can be used to subscribe to these events.

Interface ID
```
uuid( 1DC99322-69B8-4825-963A-8752D237DF5B )
```

7.1.3.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhSpdifDec_NtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

7.1.3.5.2.2 OnStandardChanged

Signature
```
[async] Void OnStandardChanged (
    [in] UInt32 cookie,
    [in] uhSpdifDec_CompStandard_t standard );
```

Qualifiers
None.

Description
This function is called by `SpdifDecAudio` whenever it detects a change in the compression standard of its digital audio input stream.

Parameters

Table 260

Name	Description
`cookie`	The value that was passed by the client at subscription time.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
standard	The new compression standard of the digital audio input stream.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

7.1.3.5.2.3 OnSampFreqChanged

Signature
```
[async] Void OnSampFreqChanged (
    [in] UInt32 cookie,
    [in] uhSpdifDec_SampFreq_t sampFreq );
```

Qualifiers
None.

Description
This function is called by SpdifDecAudio whenever it detects a change in the sampling frequency of its digital audio input stream.

Parameters

Table 261

Name	Description
cookie	The value that was passed by the client at subscription time.
sampFreq	The new sampling frequency detected by SpdifDecAudio.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

7.1.3.5.2.4 OnBitRateChanged

Signature
```
[async] Void OnBitRateChanged (
    [in] UInt32 cookie,
    [in] uhSpdifDec_BitRateInfo_t variable,
    [in] uhSpdifDec_ValStat_t     status,
    [in] UInt32                   bitRate );
```

Qualifiers
None.

Description
This function is called by `SpdifDecAudio` whenever it detects a change in the bit rate of its digital audio input stream and/or in the variability of the bit rate.

Parameters

Table 262

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`variable`	The new value indicating whether the bit rate of the digital audio input stream is variable or not.
`status`	Parameter that indicates the validity of the bit rate parameter `bitRate`.
`bitRate`	The new bit rate (in kbit/s) detected by `SpdifDecAudio`.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.3.5.2.5 OnChannelsChanged

Signature
```
[async] Void OnChannelsChanged (
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32 cookie,
    [in] uhSpdifDec_ChannelInfo_t channelInfo,
    [in] uhSpdifDec_LfeInfo_t lfePresent );
```

Qualifiers
None.

Description
This function is called by `SpdifDecAudio` whenever it detects a change in the channel configuration of its digital audio input stream and/or in the presence of a LFE channel in its digital audio input stream.

Parameters

Table 263

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`channelInfo`	The new channel configuration of the digital audio input stream.
`lfePresent`	The new value indicating the presence of a LFE channel in the digital audio input stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.3.5.2.6 OnLanguageChanged

Signature

```
[async] Void OnLanguageChanged (
    [in] UInt32 cookie,
    [in] uhSpdifDec_Language_t primaryLanguage,
    [in] uhSpdifDec_Language_t secondaryLanguage );
```

Qualifiers
None.

Description
This function is called by `SpdifDecAudio` whenever it detects a change in the primary language and/or in the secondary language of its digital audio input stream.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 264

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`primaryLanguage`	The new primary language of the digital audio input stream.
`secondaryLanguage`	The new secondary language of the digital audio input stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.3.5.2.7 OnServiceTypeChanged

Signature
```
[async] Void OnServiceTypeChanged (
    [in] UInt32 cookie,
    [in] uhSpdifDec_ServiceType_t serviceType );
```

Qualifiers
None.

Description
This function is called by `SpdifDecAudio` whenever it detects a change in the type of the audio service that the digital audio input stream conveys.

Parameters

Table 265

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`serviceType`	The new type of the audio service that the digital audio input stream conveys.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.3.5.2.8 OnError

Signature
```
[async] Void OnError (
    [in] UInt32 cookie,
    [in] uhSpdifDec_ErrorSet_t errors );
```

Qualifiers
None.

Description
This notification function is used by `SpdifDecAudio` to inform the subscribed client that one or more errors occurred when decoding audio from the audio input stream.

Parameters

Table 266

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`errors`	Describes the type(s) of error(s) that occurred.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.3.5.2.9 OnRecovered

Signature
```
[async] Void OnRecovered (
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32                cookie,
    [in] uhSpdifDec_ErrorSet_t errors );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that the SPDIF decoder has recovered from one or more previously notified errors.

Parameters

Table 267

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`errors`	Describes the type(s) of error(s) from which the SPDIF decoder has recovered.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.3.5.2.10 OnStableOutput

Signature

```
[async] Void OnStableOutput (
    [in] UInt32 cookie,
    [in] Bool stableOutput );
```

Qualifiers
None.

Description
The `SpdifDecAudio` calls this function when there is a change in the stability of the output of the `SpdifDecAudio`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 268

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`stableOutput`	If `True`: `SpdifDecAudio` is outputting decoded audio long enough so that the client can safely unmute the corresponding output at the appropriate logical component(s) downstream. If `False`: `SpdifDecAudio` is no longer outputting a stable (decoded) audio stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
True

7.1.4 UhlSpdifIn

7.1.4.1 Concepts

7.1.4.1.1 Introduction

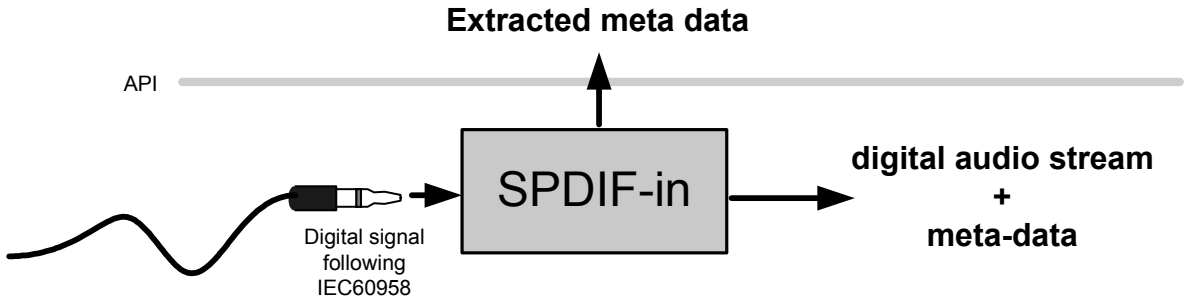


Figure 30 — Flow of data through the SPDIF-in logical component

In Figure 30 a schematic representation of the data flow through the SPDIF-in logical component is depicted. The input of the SPDIF-in logical component is a SPDIF signal that is compliant to the IEC 60958 standard [21]. Together with audio data, the SPDIF signal also contains so-called *meta-data* describing several properties of the carried audio data. The SPDIF-in logical component extracts the data from the SPDIF signal and separates the audio and meta-data. The streaming output of the SPDIF-in logical component contains audio data together with the meta-data. The extracted meta-data can be accessed by the clients of the SPDIF-in component through its interface.

The format of the audio data that is extracted from the IEC 60958 compliant signal is either linear PCM or is encoded audio. In the latter case, the extracted audio data complies to the IEC 61937 standard [19] and contains for instance AC-3, MPEG-1, MPEG-2 or DTS encoded audio. The SPDIF-in component does not *interpret* the audio data that it extracts from the SPDIF signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.1.2 Meta-data extraction

The meta-data that is carried along in the SPDIF signal is distributed over two channels. Each of the channels represents one of the so-called “sub-frames” as described by the IEC 60958 standard. Each channel has its own set of meta-data. The monitoring of meta-data of the two channels must explicitly be enabled before the meta-data is accessible to the client of the SPDIF-in logical component. Meta-data extraction of both channels can be enabled independent from each other.

The meta-data of a channel can be accessed by client by two means:

a) by means of notifications.
The properties that are part of the meta-data are divided into coherent groups. For each group one notification exists for each channel. A notification is generated when one or more properties in the meta-data belonging to the concerned group change.

b) by means of `Get<Xxx>` functions.
If monitoring of meta-data of a channel is enabled, the meta-data is continuously monitored. The most recently extracted meta-data of each channel can be retrieved by the client by means of several `Get<Xxx>` functions.

Most of the meta-data of an SPDIF signal is taken from the so-called “Channel Status” information which takes up to 24 bytes. The interpretation of the channel status information depends on its format (consumer format or professional format, see [21]). For convenience, the Channel Status bytes are interpreted and provided as grouped properties to the client. For completeness, the original Channel Status bytes can also be monitored.

7.1.4.1.3 State machine of SPDIF-in

The operation of the SPDIF-in logical component depends on its internal state. The legal states of the SPDIF-in component are the following:

— Invalid
This is the initial state of the SPDIF-in logical component. This state means that the signal information is temporarily not available.

— NotLocked
This state means that the SPDIF-in logical component has not yet recognized a signal that complies to the IEC 60958 standard. The SPDIF-in component will internally vary the sub-sample rate with which the incoming SPDIF signal is sampled.

— Locked
The SPDIF-in logical component has recognized that the incoming SPDIF signal is an IEC 60958 compliant signal. In this state, the exact audio format is not yet known. Because the syntax of the generated output stream does not depend on the audio format, streaming audio output and meta-data extraction starts once SPDIF-in is in the Locked state.
The extracted audio data may be either linear PCM or some encoded audio format. For non-linear PCM data, the SPDIF-in component tries to interpret the extracted audio data as being compliant to the IEC 61937 standard. The IEC 61937 standard describes how encoded audio formats are encoded in an IEC 60958 compliant signal.
One property of the meta-data describes whether the format of the carried audio data is linear PCM. Though, this property is not reliable. Even if the property indicates linear PCM, the SPDIF signal may contain non-linear PCM data. For this reason, even if the meta-data indicates that the carried audio is linear PCM, the SPDIF-in component will check the audio data to be compliant to the IEC 61937 standard for a fixed amount of time.

— AudioFormatFound
Once in this state, the SPDIF-in logical component has monitored the extracted audio data on IEC 61937 compliancy for a fixed amount of time. If the audio data does not comply to the IEC 61937 standard, the

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

audio format is regarded as being linear PCM. If IEC 61937 compliancy is detected, the current audio format is read from the meta-data carried in the IEC 61937 compliant audio data.

The state transition diagram of the SPDIF-in logical component is depicted in Figure 31.

Each state transition will cause a `StateChanged` notification. The SPDIF-in logical component will extract meta-data and an audio stream will be streamed out in the *Locked* state and `AudioFormatFound` state.The audio format is only known if the SPDIF-in component operates in the `AudioFormatFound` state.

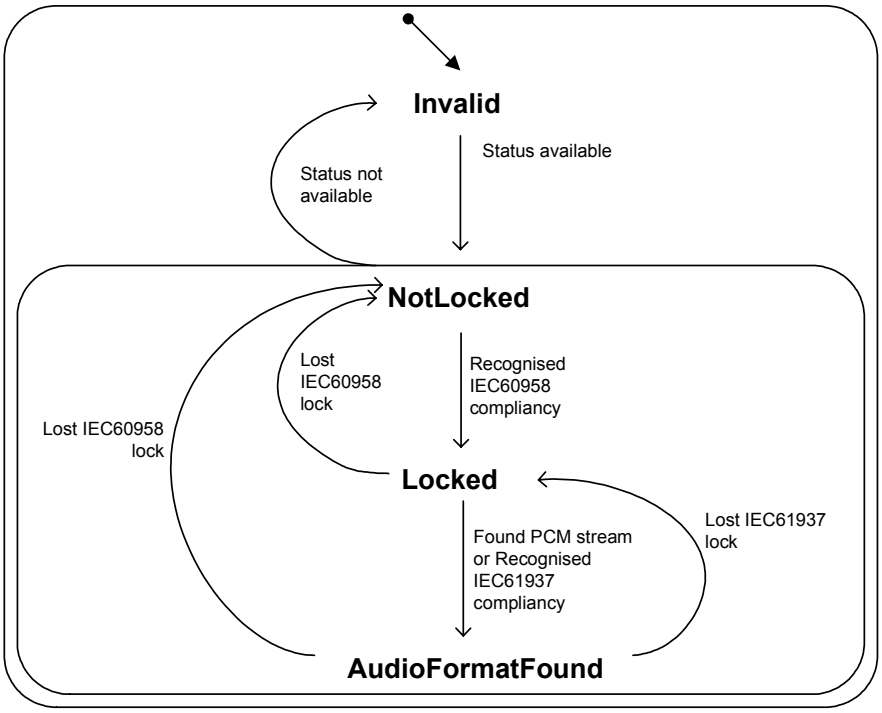


Figure 31 — State transition diagram of SPDIF-in

Special attention is needed for the case that the incoming SPDIF signal is of bad quality. Bad signal might result in high frequent state transitions. Especially transitions between the `NotLocked` and the `Locked` state may occur in a high frequency. This behavior is highly unwanted because in such situation the client would be "attacked" by an enormous amount of notifications. SPDIF-in will prevent high rate state transition notification from happening by taking a notification-threshold `notifThreshold` into account. If a state transition happens, a notification about this state change is deferred until the new state is stable for the period defined by attribute `notifThreshold`. This threshold is applied to the following transitions:

— Transition from `AudioFormatFound` state to the `Locked` or `NotLocked` state

— Transition from the `Locked` state to the `NotLocked` state.

Notifications for the transitions that are named above are generated when the new state is stable for the specified threshold. Other transitions are reported right-away. This is depicted in Figure 32.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

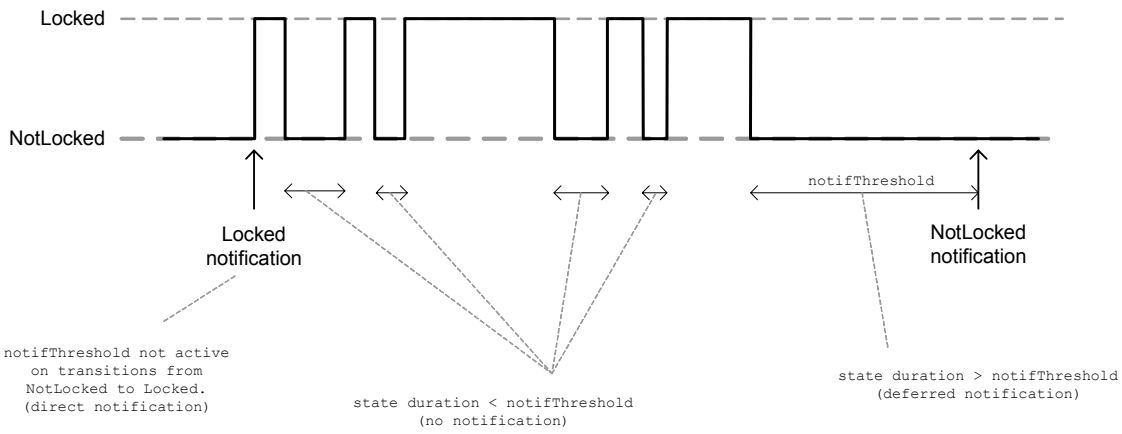


Figure 32 — example of the notification threshold for state transitions.

7.1.4.2 Types & Constants

7.1.4.2.1 Public Types & Constants

7.1.4.2.1.1 Error Codes

Signature

```
const uhErrorCode_t UH_ERR_SPDIFIN_NO_CHANNEL_DATA = 0x00000801;
```

Qualifiers

— error-codes

Description

Definition of all non-standard errors that might be returned by any of the functions of the interfaces of the SPDIF-in logical component.

Constants

Table 269

Name	Description
UH_ERR_SPDIFIN_NO_CHANNEL_DATA	This error is specific to the SPDIF-in logical component and is only used in its interface uhISpdifIn. The error signals that an attempt was made to access meta-data of a channel while no data was yet extracted from the SPDIF signal.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.2.1.2 uhSpdifIn_ValStat_t

Signature

```
typedef enum _uhSpdifIn_ValStat_t {
    uhSpdifIn_ValStatInvalid = -2,
    uhSpdifIn_ValStatUnknown = -1,
    uhSpdifIn_ValStatKnown   =  1
} uhSpdifIn_ValStat_t, *puhSpdifIn_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 270

Name	Description
uhSpdifIn_ValStatInvalid	The property/properties are temporarily not available.
uhSpdifIn_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhSpdifIn_ValStatKnown	The property/properties are available and determined.

7.1.4.2.1.3 uhSpdifIn_NrChannelStatusBytes

Signature

```
const UInt8 uhSpdifIn_NrChannelStatusBytes = 24;
```

Qualifiers
None.

Description
Defines the maximum number of bytes that is occupied by the channel status information. The consumer channel status format occupies less bytes.

7.1.4.2.1.4 uhSpdifIn_State_t

Signature

```
typedef enum _uhSpdifIn_State_t {
    uhSpdifIn_StInvalid          = -2,
    uhSpdifIn_StNotLocked        =  0,
    uhSpdifIn_StLocked           =  1,
    uhSpdifIn_StAudioFormatFound =  2
} uhSpdifIn_State_t, *puhSpdifIn_State_t;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This type enumerates the states in which the `SpdifIn` logical component may operate. See Figure 31 for the state diagram of the `SpdifIn` component.

Values

Table 271

Name	Description
`uhSpdifIn_StInvalid`	The property/properties are temporarily not available.
`uhSpdifIn_StNotLocked`	The SpdifIn component has not yet recognized an IEC 60958 compliant stream in the input signal.
`uhSpdifIn_StLocked`	The SpdifIn component has recognized an IEC 60958 compliant stream and starts outputting a digital audio stream. Several meta-data properties are available once operating in this state.
`uhSpdifIn_StAudioFormatFound`	The SpdifIn component has recognized the format of the audio data that is carried by the input signal.

7.1.4.2.1.5 uhSpdifIn_Notification_t

Signature
```
typedef enum _uhSpdifIn_Notification_t {
    uhSpdifIn_SubscriptionChanged  = 0x00000001,
    uhSpdifIn_StateChanged         = 0x00000002,
    uhSpdifIn_Error                = 0x00000004,
    uhSpdifIn_AudioInfoChanged     = 0x00000008,
    uhSpdifIn_ChannelInfoChanged   = 0x00000010,
    uhSpdifIn_CopyrightInfoChanged = 0x00000020,
    uhSpdifIn_ClockAccuracyChanged = 0x00000040,
    uhSpdifIn_ChannelStatusChanged = 0x00000080,
    uhSpdifIn_UserData             = 0x00000100
} uhSpdifIn_Notification_t, *puhSpdifIn_Notification_t;
```

Qualifiers
Enum-element.

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhISpdifInNtf`.

Values

Table 272

Name	Description
`uhSpdifIn_SubscriptionChanged`	Represents OnSubscriptionChanged.
`uhSpdifIn_StateChanged`	The operational state of the SpdifIn component changed.
`uhSpdifIn_Error`	Some error occurred while monitoring the incoming signal. The exact error that occurred will be an argument of the notification that is generated. (See type uhSpdifIn_Error_t)

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhSpdifIn_AudioInfoChanged	One or more of the properties w.r.t. the (audio) format of the SPDIF signal changed.
uhSpdifIn_ChannelInfoChanged	One or more of the properties w.r.t. the channel information of the SPDIF signal changed.
uhSpdifIn_CopyrightInfoChanged	The copyright state of the audio data carried in the SPDIF signal changed.
uhSpdifIn_ClockAccuracyChanged	The clock accuracy of the encoded clock in the SPDIF signal changed.
uhSpdifIn_ChannelStatusChanged	Any of the properties of the Channel Status information carried in the SPDIF signal changed. The Channel Status is the raw meta-data that other kind of information is extracted from.
uhSpdifIn_UserData	Userdata was extracted from the SPDIF signal.

7.1.4.2.1.6 uhSpdifIn_Notifications_t

Signature
```
typedef Int32 uhSpdifIn_Notifications_t,
              *puhSpdifIn_Notifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`SpdifIn_Notification_t`. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

7.1.4.2.1.7 uhSpdifIn_AllNotifications

Signature
```
const uhSpdifIn_Notifications_t
      uhSpdifIn_AllNotifications = 0x000001FF;
```

Qualifiers
None.

Description
Defines the set of all values of type uh`SpdifIn_Notification_t`. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

7.1.4.2.1.8 uhSpdifIn_ChannelStatusFormat_t

Signature
```
typedef enum _uhSpdifIn_ChannelStatusFormat_t {
    uhSpdifIn_CfConsumer     = 0,
    uhSpdifIn_CfProfessional = 1
} uhSpdifIn_ChannelStatusFormat_t, *puhSpdifIn_ChannelStatusFormat_t;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The uh`SpdifIn_ChannelStatusFormat_t` type enumerates the two distinct SPDIF Channel Status formats. The format is either consumer or professional, see [21].

Values

Table 273

Name	Description
`uhSpdifIn_CfConsumer`	The SPDIF channel status data format is of the consumer type.
`uhSpdifIn_CfProfessional`	The SPDIF channel status data format is of the professional type.

7.1.4.2.1.9 uhSpdifIn_ChannelNumber_t

Signature
```
typedef enum _uhSpdifIn_ChannelNumber_t {
    uhSpdifIn_Channel1    = 1,
    uhSpdifIn_Channel2    = 2
} uhSpdifIn_ChannelNumber_t, *puhSpdifIn_ChannelNumber_t;
```

Qualifiers
— enum-element.

Description
The uh`SpdifIn_ChannelNumber_t` type enumerates the two channels which are carried in the so-called “sub-frames” of a SPDIF signal. See [21] for details about sub-frames.

Values

Table 274

Name	Description
`uhSpdifIn_Channel1`	Specifies the first channel (sub-frame)
`uhSpdifIn_Channel2`	Specifies the second channel (sub-frame)

Remarks
— Note that the channel number does not *necessarily* specify e.g. the left and right channels of a stereo audio signal. AC-3, for instance, distinguishes 5 audio channels for surround sound. For audio formats other than linear PCM, the channel number is -in common- irrelevant because the two sub-frames are merged into one audio stream. In that case the meta-data of channel 1 and 2 are the same (see [21]).

7.1.4.2.1.10 uhSpdifIn_ChannelNumberSet_t

Signature
```
typedef Int32 uhSpdifIn_ChannelNumberSet_t,
              *puhSpdifIn_ChannelNumberSet_t;
```

Qualifiers
— enum-set

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The uhSpdifIn_ChannelNumberSet_t type describes a set of channel number values as defined by uhSpdifIn_ChannelNumber_t.

7.1.4.2.1.11 uhSpdifIn_AudioFormat_t

Signature

```
typedef enum _uhSpdifIn_AudioFormat_t
{
    uhSpdifIn_AfNull           = 0,
    uhSpdifIn_AfAc3            = 1,
    uhSpdifIn_AfPause          = 3,
    uhSpdifIn_AfMpeg1Layer1    = 4,
    uhSpdifIn_AfMpeg1Layer2Or3 = 5,
    uhSpdifIn_AfMpeg2          = 6,
    uhSpdifIn_AfMpeg2Aac       = 7,
    uhSpdifIn_AfMpeg2Layer1    = 8,
    uhSpdifIn_AfMpeg2Layer2Or3 = 9,
    uhSpdifIn_AfDtsType1       = 11,
    uhSpdifIn_AfDtsType2       = 12,
    uhSpdifIn_AfDtsType3       = 13,
    uhSpdifIn_AfAtrac1         = 14,
    uhSpdifIn_AfAtrac2Or3      = 15,
    uhSpdifIn_AfUnknown        = 32,
    uhSpdifIn_AfPcm            = 33
} uhSpdifIn_AudioFormat_t, *puhSpdifIn_AudioFormat_t;
```

Qualifiers
None.

Description
The uhSpdifIn_AudioFormat_t type is used to specify the format of the audio data that is carried in the SPDIF signal. The values that the audio formats represent are taken from the IEC 60958 standard.

Values

Table 275

Name	Description
uhSpdifIn_AfNull	The Null audio format is used in transitions from encoded audio (conform IEC 61937) to linear PCM data (conform IEC 60958). Rarely used.
uhSpdifIn_AfAc3	The SPDIF signal contains AC-3 encoded audio data.
uhSpdifIn_AfPause	The SPDIF signal contains a gap. During a gap, all audio is muted.
uhSpdifIn_AfMpeg1Layer1	The SPDIF signal contains MPEG-1 layer 1 encoded audio data.
uhSpdifIn_AfMpeg1Layer2Or3	The SPDIF signal contains MPEG-1 layer 2 or 3 encoded audio data or contains MPEG-2 without extension.
uhSpdifIn_AfMpeg2	The SPDIF signal contains MPEG-2 extension audio data.
uhSpdifIn_AfMpeg2Aac	The SPDIF signal contains MPEG-2 AAC encoded audio data.
uhSpdifIn_AfMpeg2Layer1	The SPDIF signal contains MPEG-2 layer 1 encoded audio data with a low sampling frequency.
uhSpdifIn_AfMpeg2Layer2Or3	The SPDIF signal contains MPEG-2 layer 2 or 3 encoded audio data with a low sampling frequency.
uhSpdifIn_AfDtsType1	The SPDIF signal contains DTS type I audio data.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhSpdifIn_AfDtsType2	The SPDIF signal contains DTS type II audio data.
uhSpdifIn_AfDtsType3	The SPDIF signal contains DTS type III audio data.
uhSpdifIn_AfAtrac1	The SPDIF signal contains ATRAC 1 encoded audio data.
uhSpdifIn_AfAtrac2Or3	The SPDIF signal contains ATRAC 2 or 3 encoded audio data.
uhSpdifIn_AfUnknown	The audio format that is carried in the SPDIF signal is not yet recognized. This value will be reported by the interface if the state of the SPDIF-in component is not “AudioFormatFound”.
uhSpdifIn_AfPcm	The audio format is linear PCM conform IEC 60958.

7.1.4.2.1.12 uhSpdifIn_Error_t

Signature

```
typedef enum _uhSpdifIn_Error_t
{
    uhSpdifIn_ErParity = 0x01,
    uhSpdifIn_ErData   = 0x02,
    uhSpdifIn_ErCrc    = 0x04
} uhSpdifIn_Error_t, *puhSpdifIn_Error_t;
```

Qualifiers

— enum-element

Description

The uhSpdifIn_Error_t type is used in notifications of type uhSpdifIn_Error to specify the type of error that occurred. All error values are represented by one bit. Multiple errors may be ORed together into one value of type uhSpdifIn_ErrorSet_t.

Values

Table 276

Name	Description
uhSpdifIn_ErParity	A parity error was detected on the IEC 60958 level. Lots of parity errors indicate a bad signal.
uhSpdifIn_ErData	The signal contains invalid data. In this case, the output audio stream might contain invalid samples.
uhSpdifIn_ErCrc	In the professional channel status format, a crc-code is provided in order to check data integrity on the IEC 60958 level. If the crc check does not match the provided crc code, data errors on the input stream exist. In that case, the output audio stream might contain invalid samples. This error can occur in professional mode only.

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.2.1.13 uhSpdifIn_ErrorSet_t

Signature

```
typedef Int32 uhSpdifIn_ErrorSet_t,
              *puhSpdifIn_ErrorSet_t;
```

Qualifiers

— enum-set

Description

The uh`SpdifIn_ErrorSet_t` type describes a set of error values as defined by uh`SpdifIn_Error_t`.

7.1.4.2.1.14 uhSpdifIn_Emphasis_t

Signature

```
typedef enum _uhSpdifIn_Emphasis_t
{
    uhSpdifIn_EmNotIndicated = 0,
    uhSpdifIn_EmNone         = 1,
    uhSpdifIn_EmCdType       = 3,
    uhSpdifIn_EmJ17          = 7
} uhSpdifIn_Emphasis_t, *puhSpdifIn_Emphasis_t;
```

Qualifiers

None.

Description

The uh`SpdifIn_Emphasis_t` type signals whether -and what kind of- pre-emphasis was applied during encoding of the audio data contained in the SPDIF signal. The values of the enumeration are derived from the IEC 60958 standard.

Values

Table 277

Name	Description
`uhSpdifIn_EmNotIndicated`	The emphasis information was not indicated in the SPDIF signal.
`uhSpdifIn_EmNone`	No pre-emphasis was applied during encoding of the audio signal.
`uhSpdifIn_EmCdType`	Pre-emphasis of type ‘CD’ was applied during encoding of the audio signal.
`uhSpdifIn_EmJ17`	Pre-emphasis of type ‘J17’ was applied during encoding of the audio signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.2.1.15 uhSpdifIn_Category_t

Signature

```
typedef enum _uhSpdifIn_Category_t
{
    uhSpdifIn_CtUnknown        = 0x80,
    uhSpdifIn_CtCdPlayer       = 0x81,
    uhSpdifIn_CtDatPlayer      = 0x83,
    uhSpdifIn_CtDccPlayer      = 0xC3,
    uhSpdifIn_CtMiniDiscPlayer = 0xC9,
    uhSpdifIn_CtDvdPlayer      = 0x99,
    // groups of products (removed high bits, 3 low-bits remaining)
    uhSpdifIn_CtLaserOptical   = 0x01,
    uhSpdifIn_CtSignalProc     = 0x02,
    uhSpdifIn_CtMagneticMedia  = 0x03,
    uhSpdifIn_CtBroadcasted    = 0x04,
    uhSpdifIn_CtMusicalInstr   = 0x05,
    uhSpdifIn_CtAdConverter    = 0x06
} uhSpdifIn_Category_t, *puhSpdifIn_Category_t;
```

Qualifiers
None.

Description
The category code indicates the kind of equipment that generated the digital audio signal. According to IEC 60958 the list of categories is not fixed and may be extended in the future. Though, the IEC 60958 standard distinguishes category *groups* that are fixed. The category enumeration describes both, specific equipment and grouped equipment.

The category code is only known if the channel status data that is extracted from the SPDIF signal is of the "consumer" type. In case the channel status is of the "professional" type, the category code is uh`SpdifIn_CtUnknown`.

Values

Table 278

Name	Description
`uhSpdifIn_CtUnknown`	The category is not specified.
`uhSpdifIn_CtCdPlayer`	The audio signal is generated by a CD player.
`uhSpdifIn_CtDatPlayer`	The audio signal is generated by a DAT player.
`uhSpdifIn_CtDccPlayer`	The audio signal is generated by a DCC player.
`uhSpdifIn_CtMiniDiscPlayer`	The audio signal is generated by a MiniDisc player.
`uhSpdifIn_CtDvdPlayer`	The audio signal is generated by a DVD player.
`uhSpdifIn_CtLaserOptical`	The generating device belongs to the optical laser equipment group. This is a grouped category.
`uhSpdifIn_CtSignalProc`	The generating device is any kind of signal processing device. This is a grouped category.
`uhSpdifIn_CtMagneticMedia`	The generating device belongs to the magnetic tape or magnetic disc product group. This is a grouped category.
`uhSpdifIn_CtBroadcasted`	Audio signal is broadcast. This is a grouped category.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhSpdifIn_CtMusicalInstr	Audio signal is generated by any kind of musical instrument. This is a grouped category.
uhSpdifIn_CtAdConverter	Audio signal is generated by any kind of A/D converter. This is a grouped category.

Remarks

— The last 6 category codes specify a *group* of devices rather than one *specific* device.

7.1.4.2.1.16 uhSpdifIn_ClockAccuracy_t

Signature

```
typedef enum _uhSpdifIn_ClockAccuracy_t
{
    uhSpdifIn_CaInvalid   = -2,
    uhSpdifIn_CaUnknown   = -1,
    uhSpdifIn_Ca1000ppm   =  0,
    uhSpdifIn_Ca50ppm     =  1,
    uhSpdifIn_CaVariable  =  2
} uhSpdifIn_ClockAccuracy_t, *puhSpdifIn_ClockAccuracy_t;
```

Qualifiers

None.

Description

The uh`SpdifIn_ClockAccuracy_t` type is used to specify the accuracy of the clock that is sent along with the SPDIF signal.

Values

Table 279

Name	Description
uhSpdifIn_CaInvalid	The clock accuracy is temporarily not available.
uhSpdifIn_CaUnknown	The clock accuracy is available, yet it could not be determined.
uhSpdifIn_Ca1000ppm	The tolerance of transmitting sample frequency shall be within +/- 1000 x 10-6.
uhSpdifIn_Ca50ppm	The tolerance of transmitting sample frequency shall be within +/- 50 x 10-6.
uhSpdifIn_CaVariable	The clock-accuracy may vary in time. The signal in this mode can only be received by specially designed receivers. Most receivers require a constant time base in the SPDIF signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.2.1.17 uhSpdifIn_CopyrightStatus_t

Signature

```
typedef enum _uhSpdifIn_CopyrightStatus_t
{
    uhSpdifIn_CrInvalid     = -2,
    uhSpdifIn_CrUnknown     = -1,
    uhSpdifIn_CrNone        =  1,
    uhSpdifIn_CrPreRecorded =  2,
    uhSpdifIn_CrHomeCopy    =  3
} uhSpdifIn_CopyrightStatus_t, *puhSpdifIn_CopyrightStatus_t;
```

Qualifiers
None.

Description
The uh`SpdifIn_CopyrightStatus_t` type is used to signal the copyright status of the input signal. This type combines 2 copyright properties into one enumeration: the copyright property (copyrighted or not) and the generation status (number of copies allowed) of the data.

If a signal carries copyrighted material, the copyright status is either uh`SpdifIn_CrPreRecorded` or uh`SpdifIn_CrHomeCopy`. If the audio signal is not copyrighted, the copyright state is uh`SpdifIn_CrNone`.

Values

Table 280

Name	Description
`uhSpdifIn_CrInvalid`	The copyright information is temporarily not available.
`uhSpdifIn_CrUnknown`	The copyright information is available, yet it could not be determined. (always the case for the professional format).
`uhSpdifIn_CrNone`	The audio data carried in the SPDIF signal is not copyrighted.
`uhSpdifIn_CrPreRecorded`	The audio data carried in the SPDIF signal is copyrighted. One more copy is allowed.
`uhSpdifIn_CrHomeCopy`	The audio data carried in the SPDIF signal is copyrighted. Copying is not allowed.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.2.2 Model Types & Constants

7.1.4.2.2.1 ConsumerChannelStatus_t

Signature

```
typedef struct _ConsumerChannelStatus_t
{
    int channelStatusFormat : 1;
    int nonLinearPcm        : 1;
    int copyright           : 1;
    int emphasis            : 3;
    int channelStatusMode   : 2;
    int categoryCode        : 8;
    int sourceNumber        : 4;
    int channelNumber       : 4;
    int sampleFrequency     : 4;
    int clockAccuracy       : 2;
    int reserved1           : 2;
    int wordLengthFieldSize : 1;
    int wordLengthBits      : 3;
    int reserved2           : 4;
} ConsumerChannelStatus_t, *pConsumerChannelStatus_t;
```

Qualifiers

None.

Description

The `ConsumerChannelStatus_t` type represents the Channel Status bits as described in the IEC 60958 standard (see [21]).

Fields

Table 281

bits	Name	Description
0	`channelStatusFormat`	0 = Consumer format
		1 = Professional format
1	`nonLinearPcm`	0 = Linear PCM audio signal
		1 = Other than linear PCM samples
2	`copyright`	0 = Copyrighted material. The L bit of the categoryCode field defines the copyright generation status of the channel.
		1 = Not copyrighted
3-5	`emphasis`	000 = Emphasis not indicated
		001 = Emphasis CD type
6-7	`channelStatusMode`	Should always contain ‘0’, all other values reserved.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

bits	Name	Description
8-15	categoryCode	Defines the type of equipment that generated the SPDIF signal. The MSB of this field ('L') describes the copyright generation status of the channel. Most common values are: 0000 0001 = CD player L000 0011 = DAT player L100 0011 = DCC player L100 1001 = MiniDisc player L001 1001 = DVD player
16-19	sourceNumber	Contains the source number (0..15)
20-23	channelNumber	Contains the channel number (0..15)
24-27	sampleFrequency	0000 = 44.1 kHz 0010 = 48.0 kHz 0011 = 32.0 kHz
28-29	clockAccuracy	Defines the tolerance of the sampling frequency of the transmitter. 01 = Level I (±50 parts per million) 00 = Level II (±1000 parts per million) 10 = Level III (variable pitch shifted)
30-31	reserved1	Not used, reserved for future extensions
32	wordLengthFieldSize	Together with field wordLengthBits defines the number of bits in each audio sample. 0 = Maximum length of an audio sample is 20 bits 1 = Maximum length of an audio sample is 24 bits
33-35	wordLengthBits	Together with field wordLengthFieldSize defines the number of bits in each audio sample. Value of field wordLengthFieldSize: 1: 0: --- --- 000 Not indicated Not indicated 101 24 bits 20 bits 100 23 bits 19 bits 010 22 bits 18 bits 110 21 bits 17 bits 001 20 bits 16 bits
36-39	reserved2	Not used, reserved for future extensions.

Constraints

— The channelStatusFormat field should always be set to '0'.

Remarks

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Bit-fields are not portable due to the fact that the alignment may vary between compilers. Though, the `ConsumerChannelStatus_t` type is only a model type and won't be used in the implementation of the SPDIF-in component. For this document 8-bits alignment is assumed for bit-fields.

7.1.4.2.2.2 ProfessionalChannelStatus_t

Signature

```
typedef struct _ProfessionalChannelStatus_t
{
    int channelStatusFormat    : 1;
    int nonLinearPcm           : 1;
    int emphasis               : 3;
    int unused1                : 1;
    int sampleFrequency1       : 2;
    int unused2                : 4;
    int unused3                : 4;
    int useOfAuxWord           : 3;
    int wordLengthBits         : 3;
    int unused4                : 2;
    int channelId              : 8;
    int unused5                : 2;
    int reserved1              : 1;
    int sampleFrequency2       : 4;
    int sampleFrequencyScaling : 1;
    int unused6                : 32;
    int unused7                : 32;
    int unused8                : 32;
    int unused9                : 32;
    int reliabilityFlags       : 8;
    int crcc                   : 8;
} ProfessionalChannelStatus_t, *pProfessionalChannelStatus_t;
```

Qualifiers
None.

Description
The `ProfessionalChannelStatus_t` type represents the Channel Status bits as described in the IEC 60958 standard (see [21]).

Fields

Table 282

bits	Name	Description
0	`channelStatusFormat`	0 = consumer format
		1 = professional format
1	`nonLinearPcm`	0 = linear PCM audio signal
		1 = other than linear PCM samples
2-4	`emphasis`	000 = Emphasis not indicated
		001 = No emphasis
		011 = CD type emphasis
		111 = J-17 emphasis
5	`unused1`	Contains the lock field, but is not used by SpdifIn.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

bits	Name	Description
6-7	sampleFrequency1	00 = not indicated (see bits 35-38 for the sample rate) 01 = 48.0 kHz 10 = 44.1 kHz 11 = 32.0 kHz
8-11	unused2	Contains the channel mode field, but is not used by SpdifIn.
12-15	unused3	Contains the user bit management field, but is not used by SpdifIn.
16-18	useOfAuxWord	Defines whether the lower 4 bits of 24 bits sample data belong to the sample value or contains auxiliary data. Together with the wordLengthBits defines the length of the sample words. 000 = not defined, audio max 20 bits 100 = used for main audio, max 24 bits 010 = used for coordination signal, audio max 20 bits 110 = user defined.
19-21	wordLengthBits	Depending on field useOfAuxWord: if max==20 bits / if max==24 bits 000 not indicated / not indicated 101 24 bits / 20 bits 100 23 bits / 19 bits 010 22 bits / 18 bits 110 21 bits / 17 bits 001 20 bits / 16 bits
22-23	unused4	Contains the alignment level field, but is not used by SpdifIn.
24-31	channelId	If the highest bit == 0 then the channel number is 1 plus the numeric value of the lower 7 bits. If the highest bit == 1 then bit 4-6 define a multi channel mode and bits 0-3 give the channel number within that mode.
32-33	unused5	Contains the DARS field, but is not used by SpdifIn.
34	reserved1	Reserved for future extensions.
35-38	sampleFrequency2	Extension to the sample rates provided by bit 6-7. Only to be used if bit 6-7 contain '00'. 0000 = not indicated 0001 = 24.00 kHz 0010 = 96.00 kHz 1001 = 22.05 kHz 1010 = 88.20 kHz 1011 = 176.40 kHz 1111 = user defined
39	sampleFrequencyScaling	0 = no scaling 1 = apply factor of 1/1.001 to the value defined by field sampleFrequency2

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

bits	Name	Description
48-79	unused6	Contains the channel origin field, but is not used by SpdifIn.
80-111	unused7	Contains the channel destination field, but is not used by SpdifIn.
112-143	unused8	Contains the sample address code field, but is not used by SpdifIn.
144-175	unused9	Contains the time of day code field, but is not used by SpdifIn.
176-183	reliabilityFlags	Defines whether the content of the channel status bytes is reliable. This field is not used by the SPDIF-in component.
184-191	crcc	Provides a CRC code for data integrity check purposes. 00000000 no CRC code provided xxxxxxxx CRC code for all fields in the channel status up to this field.

Constraints

— The channelStatusFormat field should always be set to '1'.

Remarks

— Bit-fields are not portable due to the fact that the alignment may vary between compilers. Though, the ConsumerChannelStatus_t type is only a model type and won't be used in the implementation of the SPDIF-in component. For this document 8-bits alignment is assumed for bit-fields.

7.1.4.2.2.3 ChannelStatus_t

Signature

```
typedef union _ChannelStatus_t
{
    ConsumerChannelStatus_t     consumer;
    ProfessionalChannelStatus_t professional;
} ChannelStatus_t, *pChannelStatus_t;
```

Qualifiers

None.

Description

The ChannelStatus_t union is used to interpret a block of 24 bytes channel status information extracted from the SPDIF signal. The interpretation of the channel status bytes depends on the very first bit of the block bit 7 of byte 0). If this bit is 0, then the consumer member must be used, else the professional member must be used for interpretation. Bit 0 can be interpreted with both union members.

Fields

Table 283

Name	Description
consumer	Bit-oriented interpretation of the Channel Status bytes for the consumer format.
professional	Bit-oriented interpretation of the Channel Status bytes for the professional format.

Constraints

— If consumer.channelStatusFormat == 0 then only member consumer should be used.

— If consumer.channelStatusFormat == 1 then only member professional should be used.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.2.2.4 BurstInfo_t

Signature

```
typedef struct _BurstInfo_t
{
    UInt8 dataType          : 5;
    UInt8 reserved1         : 2;
    UInt8 errorFlag         : 1;
    UInt8 reserved2         : 8;
} BurstInfo_t, *pBurstInfo_t;
```

Qualifiers
None.

Description
The `BurstInfo_t` type describes the format of the burst info field of a non-linear PCM data burst compliant to the IEC 61937 standard (see [19]).

Fields

Table 284

Name	Description
`dataType`	This field indicates the format of the burst payload. The following data Types are defined: 00000 = Null data 00001 = AC-3 data 00011 = Pause 00100 = MPEG-1 layer 1 data 00101 = MPEG-1 layer 2 or 3 data, or MPEG-2 without extension 00110 = MPEG-2 data with extension 00111 = MPEG-2 AAC data 01000 = MPEG-2 layer 1 low sampling frequency 01001 = MPEG-2 layer 2 or 3 low sampling frequency 01011 = DTS level I 01100 = DTS level II 01101 = DTS level III 01110 = ATRAC 1 01111 = ATRAC 2 or 3
`reserved1`	Reserved for future extensions.
`errorFlag`	0 = no error, valid burst payload. 1 = The burst payload may contain errors.
`reserved2`	Reserved for future extensions.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.2.2.5 ChannelMetaData_t

Signature

```
typedef struct _ChannelMetaData_t
{
    ChannelStatus_t channelStatus;
    UInt8           userData[uhSpdifIn_NrChannelStatusBytes];
    BurstInfo_t     burstInfo;
} ChannelMetaData_t, *pChannelMetaData_t;
```

Qualifiers
None.

Description
The `ChannelMetaData_t` type bundles all channel related information into one structure. In this scope, a channel is a sub-frame according the IEC 60958 standard [21].

Fields

Table 285

Name	Description
`channelStatus`	Channel Status bits that are extracted from the SPDIF signal.
`userData`	User Data bytes that are extracted from the SPDIF signal.
`burstInfo`	If an IEC 61937 compliant burst is found, the burst information is collected and stored in this field.

7.1.4.3 Logical Component

7.1.4.3.1 Interface-Role Model

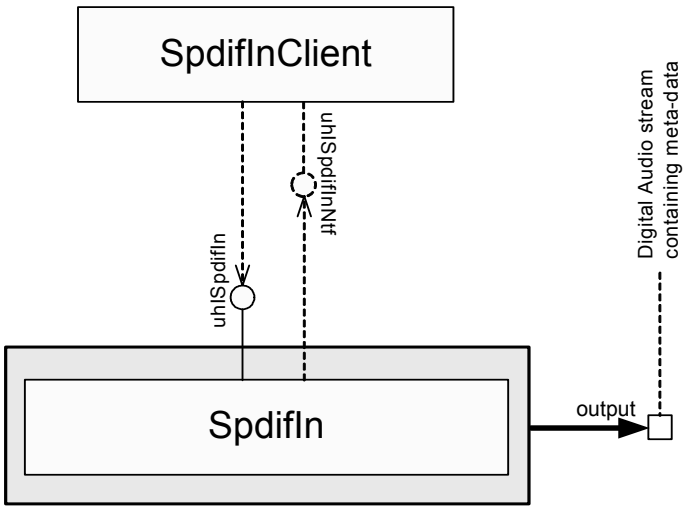


Figure 33 — Interface-Role Model of the SPDIF-in logical component

Figure 33 represents the interface role model for the SPDIF-in logical component. It shows the names and roles that the SPDIF-in logical component and its clients have. The role `SpdifIn` provides only one interface

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

to its clients: uhISpdifIn. The uhISpdifIn interface provides the clients access to meta-data that was extracted from the input SPDIF signal.

In case of events happening in the SPDIF-in component that are asynchronous to its client behavior, the SpdifIn role might generate notifications. If the client of SpdifIn provides an interface of type uhISpdifInNtf, these notifications can be received by the client.

The SpdifIn role extracts audio data and accompanying meta-data from a SPDIF input signal and streams out the extracted data along with the meta-data. The SpdifIn role has no *streaming* input.

7.1.4.3.2 Diversity

7.1.4.3.2.1 Provided Interfaces

Table 286

Role	Interface	Presence
SpdifIn	uhISpdifIn	mandatory
SpdifInClient	uhISpdifInNtf	optional

7.1.4.3.2.2 Configurable Items

Table 287

Role	Attribute
SpdifIn	Bool multiChannelSupp
SpdifIn	UInt32 notifThreshold

Not all implementations of SpdifIn may support monitoring meta-data of two channels at the same time. The multiChannelSupp attribute makes the two-channel support configurable.

7.1.4.3.3 Instantiation

7.1.4.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 288

Type	Object	Multiplicity
SpdifIn	spdifIn	1

7.1.4.3.3.2 Initial State

The initial state of a SPDIF-in logical component:

— spdifIn.state = uhSpdifIn_StInvalid
The state of SpdifIn is set to uhSpdifIn_StInvalid. Directly after creation, SpdifIn starts trying to recognize an IEC 60958 compliant signal. The component does not have to be started explicitly.

— spdifIn.monitoredChannels == 0
Initially, monitoring of both channels for meta-data extraction is disabled.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.3.4 Execution Constraints

The SPDIF-in logical component acts as an autonomous component. This means that, once instantiated, the behavior of the component does not require interaction of its client(s). The interaction of the client is restricted to monitoring meta-data that was extracted from the SPDIF input signal. This monitoring can be done either by means of notifications (`uhISpdifInNtf`) or by means of query (`GetXxx`) functions from interface `uhISpdifIn`.

7.1.4.4 Roles

7.1.4.4.1 SpdifIn

Signature

```
role SpdifIn
{
    uhSpdifIn_State_t            state;
    uhSpdifIn_ChannelNumberSet_t monitoredChannels;
    ChannelMetaData_t            channel[2];
    Bool                         multiChannelSupp
    UInt32                       notifThreshold
}
```

Qualifiers

— root

Description

This role models an instance of a SPDIF-in logical component, providing an interface of type `uhISpdifIn`.

Independent Attributes

Table 289

Name	Description
`state`	Describes the state in which the component operates.
`monitoredChannels`	Describes which channels are currently monitored for meta-data.
`channel`	Contains the meta-data of channels for which monitoring is enabled. Depending on attribute monitoredChannels either none, one or both of these meta-data structures are in use.
`multiChannelSupp`	Defines whether extraction of meta-data for two channels at the same time is supported by the implementation of the SPDIF-in component.
`notifThreshold`	Specifies the time (in milliseconds) that a new state must be stable before a notification of type uhSpdifIn_StateChanged is generated. This threshold is only applied to the following state transitions: - Transition from AudioFormatFound state to the Locked or NotLocked state - Transition from the Locked state to the NotLocked state.

Invariants

— if(`multiChannelSupp`)
  — `nrElementsInSet`[3](`monitoredChannels`) <= 2

[3]Function nrElementsInSet is an imaginary function that counts the number of selected enum-elements in a enum-set.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— else
  — `nrElementsInSet(monitoredChannels)` <= 1

Instantiation
See subclause 7.1.4.3.3.

Streaming Behavior
The SPDIF-in logical component is a so-called *digitizer*. This means that the component has no streaming input but gets SPDIF signal as input. From the SPDIF input signal, the SPDIF-in component extracts meta-data and audio data. The component has a streaming output containing the extracted audio data.

Active Behavior

Table 290

Event	Action
New SPDIF data from the SPDIF signal is fetched into so-called frames.	Update the meta-data in attribute channel, and if the client subscribed on notification(s), generate the appropriate notification(s). The extracted audio frames are output to the output pin.

Execution Constraints
None.

7.1.4.4.2 SpdifInClient

Signature
`role` `SpdifInClient {}`

Qualifiers
— actor

Description
This role models the client of the SPDIF-in logical component

Execution Constraints
None.

7.1.4.5 Interfaces

7.1.4.5.1 uhISpdifIn

Qualifiers
None.

Description
The `uhISpdifIn` interface is the only interface of a SPDIF-in logical component. Main goal of this interface is monitoring of meta-data that is extracted from a SPDIF input signal.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 2633B732-5F95-4d63-941D-7DEE9F57D67D )`

Execution Constraints
None.

7.1.4.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhISpdifInNtf*            pINotify,
    [in] UInt32                    cookie,
    [in] uhSpdifIn_Notifications_t notifs );
```

Qualifiers
— subscribe-function

7.1.4.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhISpdifInNtf*            pINotify,
    [in] UInt32                    cookie,
    [in] uhSpdifIn_Notifications_t notifs );
```

Qualifiers
— unsubscribe-function

7.1.4.5.1.3 SetMonitoredChannels

Signature
```
uhErrorCode_t SetMonitoredChannels(
    [in] uhSpdifIn_ChannelNumberSet_t monitoredChannels);
```

Qualifiers
— Synchronous (effects of this function are asynchronous)

Description
This function selects the channel(s) for which meta-data must be monitored. This function is used for both, `enabling` and `disabling` monitoring of channels. The semantics of this function is best explained by an example scenario:

— Multiple channels are enabled as follows:

```
SetMonitoredChannels( uhSpdifIn_Channel1 | uhSpdifIn_Channel2 );
```

— Next, channel 1 is disabled as follows:

```
SetMonitoredChannels( uhSpdifIn_Channel2 );
```

— Finally, monitoring of all channels is disabled:

```
SetMonitoredChannels( 0 );
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The monitoring of the meta-data of the incoming SPDIF signal happens asynchronously.

Parameters

Table 291

Name	Description
`monitoredChannels`	Specifies for which channels the monitoring must be enabled

Return Values
Standard.

Precondition
— if `GetMultiChannelSupp()` returns False, only one channel must be enabled at the same time.

Action
Modify `spdifIn.monitoredChannels`.

Postcondition
`spdifIn.monitoredChannels == monitoredChannels`

7.1.4.5.1.4 GetMonitoredChannels

Signature

```
uhErrorCode_t GetMonitoredChannels(
    [out] uhSpdifIn_ChannelNumberSet_t* pMonitoredChannels);
```

Qualifiers
— Synchronous

Description
Reports for which channels the meta-data is currently monitored.

Parameters

Table 292

Name	Description
`pMonitoredChannels`	Output parameter that describes the channels that are currently monitored.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pMonitoredChannels == spdifIn.monitoredChannels
```

7.1.4.5.1.5 GetMultiChannelSupp

Signature
```
uhErrorCode_t GetMultiChannelSupp(
    [out] Bool* pMultiChannelSupp );
```

Qualifiers
— Synchronous

Description
Queries the capability of the SPDIF-in implementation w.r.t. extraction of meta-data on two channels simultaneously.

Parameters

Table 293

Name	Description
`pMultiChannelSupp`	Output parameter that on exit will indicate if the SPDIF-in component is able to extract meta-data from two channels simultaneously.

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
*pMultiChannelSupp == multiChannelSupp
```

7.1.4.5.1.6 GetState

Signature
```
uhErrorCode_t GetState(
    [out] uhSpdifIn_State_t* pStatus );
```

Qualifiers
— Synchronous

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the current operational state of the SPDIF-in logical component.

Parameters

Table 294

Name	Description
pStatus	On function exit this parameter will indicate the current status of the component.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pStatus == spdifIn.state`

7.1.4.5.1.7 GetAudioInfo

Signature

```
uhErrorCode_t GetAudioInfo(
    [in]  uhSpdifIn_ChannelNumber_t        chNr,
    [out] uhSpdifIn_ValStat_t*             pStatus,
    [out] uhSpdifIn_ChannelStatusFormat_t* pCsFormat,
    [out] uhSpdifIn_AudioFormat_t*         pType,
    [out] uhSpdifIn_Emphasis_t*            pEmphasis,
    [out] uhSpdifIn_Category_t*            pCategory,
    [out] UInt32*                          pSamplingFrequency,
    [out] UInt8*                           pAudioSampleWordLength );
```

Qualifiers
— Synchronous

Description
The `GetAudioInfo` function returns a sub-set of the meta-data for the specified channel that describes the properties of audio data carried by the SPDIF signal. Since the meta-data is constantly updated with new meta-data that is extracted from the SPDIF signal, the returned audio information is a “snapshot”. However, a change in the audio properties of the audio data carried by the SPDIF signal only happens occasionally. Clients that provide the `uhISpdifInNtf` interface may also monitor *changes* of the audio properties by subscribing on notification uh`SpdifIn_AudioInfoChanged`.

Note that prior to use this function, the monitoring of the concerned channel should be enabled by invoking function `SetMonitoredChannels`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 295

Name	Description
chNr	Input parameter that specifies the channel for which the audio properties are requested.
pStatus	Output parameter that indicates the validity of the [out] parameters of this function.
pCsFormat	Output parameter that on function exit indicates the Channel Status format which is either uhSpdifIn_CfConsumer or uhSpdifIn_CfProfessional.
pType	On function exit this output parameter contains the audio format of the output audio stream (PCM, MPEG, AC-3 etc..).
pEmphasis	Output parameter that on function exit contains the emphasis type of the output audio stream.
pCategory	Output parameter that on function exit contains the sort of equipment that generated the SPDIF input signal.
pSamplingFrequency	Output parameter that contains the sampling frequency of the output audio stream expressed in Hz.
pAudioSampleWordLength	Output parameter that contains the number of significant bits that are used per audio sample. This value can maximum be 24.

Return Values
Standard.

Precondition
```
(spdifIn.monitoredChannels & chNr) == chNr (monitoring of meta-data of the selected channel is enabled)
```

Action
None.

Postcondition
```
— if (spdifIn.state == uhSpdifIn_StInvalid)
  — *pStatus == uhSpdifIn_ValStatInvalid
  else if ((spdifIn.state == uhSpdifIn_StNotLocked) ||
      (spdifIn.state == uhSpdifIn_StLocked))
  — *pStatus == uhSpdifIn_ValStatUnknown
  else if (spdifIn.state == uhSpdifIn_StAudioFormatFound)
  — *pStatus == uhSpdifIn_ValStatKnown
  — *pCsFormat == spdifIn.channel[chNr-1].channelStatus.consumer.channelStatusFormat
  — *pType == spdifIn.channel[chNr-1].channelStatus.consumer.nonLinearPcm ? spdifIn.channel[chNr-1].burstInfo.dataType : uhSpdifIn_AfPCM
— if (*pCsFormat == uhSpdifIn_CfConsumer)
  — *pEmphasis == spdifIn.channel[chNr-1].channelStatus.consumer. emphasis
  — *pCategory == f 4 (spdifIn.channel[chNr-1].channelStatus.consumer.categoryCode)
```

[4] Function f() represents an imaginary function that converts its input parameters into an appropriate result. The exact description of the applied processing is out of scope of this interface specification.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    — *pSamplingFrequency              ==              f(spdifIn.channel[chNr-1].
      channelStatus.consumer.sampleFrequency)
    — *pAudioSampleWordLength          ==              f(spdifIn.channel[chNr-1].
      channelStatus.consumer.wordLengthFieldSize,spdifIn.          channel[chNr-
      1].channelStatus.consumer.wordLengthBits)
— if (pCsFormat == uhSpdifIn_CfProfessional)
    — *pEmphasis == spdifIn.channel[chNr-1].channelStatus.professional. emphasis
    — *pCategory == uhSpdifIn_CtUnknown
    — *pSamplingFrequency              ==              f(spdifIn.channel[chNr-
      1].channelStatus.professional.sampleFrequency1,      spdifIn.channel[chNr-
      1].channelStatus.professional.sampleFrequency2,      spdifIn.channel[chNr-
      1].channelStatus.professional. sampleFrequencyScaling)
    — *pAudioSampleWordLength          ==              f(spdifIn.channel[chNr-1].
      channelStatus.professional.useOfAuxWord,             spdifIn.channel[chNr-
      1].channelStatus.professional.wordLengthBits)
```

7.1.4.5.1.8 GetChannelInfo

Signature

```
uhErrorCode_t GetChannelInfo(
    [in]  uhSpdifIn_ChannelNumber_t      chNr,
    [out] uhSpdifIn_ValStat_t*           pStatus,
    [out] UInt8*                         pChannelStatusMode,
    [out] UInt8*                         pSourceNumber,
    [out] UInt8*                         pChannelNumber);
```

Qualifiers

— Synchronous

Description

The `GetChannelInfo` function can be used to request channel related information. This channel information is part of the meta-data that belongs to a channel.

Clients that provide the `uhISpdifInNtf` interface may also monitor *changes* of the channel information by subscribing on notification uh`SpdifIn_ChannelInfoChanged`.

Note that prior to use this function, the monitoring of the concerned channel should be enabled by invoking function `SetMonitoredChannels`.

Parameters

Table 296

Name	Description
`chNr`	Input parameter that specifies the channel for which the channel information is requested.
`pStatus`	Output parameter that indicates the validity of the [out] parameters of this function.
`pChannelStatusMode`	Output parameter that on function exit indicates the Channel Mode Status field. The returned value should always be '0'.
`pSourceNumber`	Output parameter that on function exit contains the source number for the carried audio.
`pChannelNumber`	Output parameter that on function exit contains the *logical* channel number for the carried audio.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
(spdifIn.monitoredChannels & chNr) == chNr (monitoring of meta-data of the selected channel is enabled)

Action
None.

Postcondition
— if (spdifIn.state == uhSpdifIn_StInvalid)
  — *pStatus == uhSpdifIn_ValStatInvalid
  else if ((spdifIn.state == uhSpdifIn_StNotLocked) ||
      (spdifIn.state == uhSpdifIn_StLocked))

  — *pStatus == uhSpdifIn_ValStatUnknown
  else if (spdifIn.state == uhSpdifIn_StAudioFormatFound)

  — *pStatus == uhSpdifIn_ValStatKnown
— if (spdifIn.channel[chNr-1].consumer.channelStatusFormat == uhSpdifIn_CfConsumer)
  — *pChannelStatusMode == spdifIn.channel[chNr-1].channelStatus.consumer.channelStatusMode
  — *pSourceNumber == spdifIn.channel[chNr-1].channelStatus.consumer.sourceNumber
  — *pChannelNumber == spdifIn.channel[chNr-1].channelStatus.consumer.channelNumber
— if (spdifIn.channel[chNr-1].consumer.channelStatusFormat == uhSpdifIn_CfProfessional)
  — *pChannelStatusMode == 0
  — *pSourceNumber == 0
  — *pChannelNumber == f5(spdifIn.channel[chNr-1].channelStatus.professional.channelId)

7.1.4.5.1.9 GetCopyrightInfo

Signature
```
uhErrorCode_t GetCopyrightInfo(
    [in]  uhSpdifIn_ChannelNumber_t    chNr,
    [out] uhSpdifIn_CopyrightStatus_t* pCopyrightStatus);
```

Qualifiers
— Synchronous

[5]Function f() represents an imaginary function that combines its input parameters into a result. The exact description of the applied processing is out of scope of this interface specification.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The `GetCopyrightInfo` function returns the current copyright status of the audio data in the SPDIF signal for a given channel.

Parameters

Table 297

Name	Description
`chNr`	Input parameter specifying the channel for which the copyright information is requested.
`pCopyrightStatus`	On function exit this output parameter will indicate the current copyright status of the audio data in the SPDIF signal.

Return Values
Standard.

Precondition
`(spdifIn.monitoredChannels & chNr) == chNr` (monitoring of meta-data of the selected channel is enabled)

Action
None.

Postcondition

```
— if (spdifIn.state == uhSpdifIn_StInvalid)
   —  *pClockAccuracy == uhSpdifIn_CaInvalid
   else
   — if (spdifIn.channel[chNr-1].consumer.channelStatusFormat ==
     uhSpdifIn_CfConsumer)
     — *pCopyrightStatus ==
       spdifIn.channel[chNr1].channelStatus.consumer.copyright ?
       uhSpdifIn_CrNone : f[6](spdifIn.channel[chNr-1].channelStatus.
       consumer.categoryCode)

   — if (spdifIn.channel[chNr-1].professional. channelStatusFormat ==
     uhSpdifIn_CfProfessional)
     — *pCopyrightStatus == uhSpdifIn_CrUnknown
```

[6] Function f() represents an imaginary function that combines its input parameters into a result. The exact description of the applied processing is out of scope of this interface specification.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.5.1.10 GetClockAccuracy

Signature

```
uhErrorCode_t GetClockAccuracy(
    [in]  uhSpdifIn_ChannelNumber_t  chNr,
    [out] uhSpdifIn_ClockAccuracy_t* pClockAccuracy );
```

Qualifiers

— Synchronous

Description

This function can be used to query the accuracy of the encoded clock in the SPDIF signal.

Parameters

Table 298

Name	Description
`chNr`	Input parameter specifying the channel for which the clock accuracy information is requested.
`pClockAccuracy`	This output parameter will on function exit indicate the clock accuracy.

Return Values

Standard.

Precondition

— `(spdifIn.monitoredChannels & chNr) == chNr` (monitoring of meta-data of the selected channel is enabled)

Action

None.

Postcondition

— if `(spdifIn.state == uhSpdifIn_StInvalid)`
  — `*pClockAccuracy == uhSpdifIn_CaInvalid`
  else
  — `if (spdifIn.channel[chNr-1].consumer.channelStatusFormat == uhSpdifIn_CfConsumer)`
    — `*pClockAccuracy == spdifIn.channel[chNr-1].consumer.clockAccuracy`
  — `if (spdifIn.channel[chNr-1].professional.channelStatusFormat == uhSpdifIn_CfProfessional)`
    — `*pClockAccuracy == uhSpdifIn_CaUnknown`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.5.1.11 GetChannelStatusBytes

Signature

```
uhErrorCode_t GetChannelStatusBytes(
    [in]      uhSpdifIn_ChannelNumber_t chNr,
    [out]     uhSpdifIn_ValStat_t*      pStatus,
    [in, out] UInt8                     pData[uhSpdifIn_NrChannelStatusBytes]
 );
```

Qualifiers

— Synchronous

Description

The `GetChannelStatusBytes` function can be used to get the "raw" channel status bytes as extracted from the SPDIF signal. The data is returned as an array of bytes. The client itself is responsible for the interpretation of the data. The format of the channel status data can be found in the IEC 60958 standard.

Parameters

Table 299

Name	Description
`chNr`	Input parameter specifying the channel for which the channel status bytes are requested.
`pStatus`	Output parameter that indicates the validity of the "raw" channel status bytes.
`pData`	Input / Output parameter that refers to the "raw" channel status bytes. The client is responsible to pass a valid pointer to a uh`SpdifIn_NrChannelStatusBytes` bytes memory block. The channel status information will be copied into the memory block that is provided by the client.

Return Values

Standard

Precondition

— `(spdifIn.monitoredChannels & chNr) == chNr` (monitoring of meta-data of the selected channel is enabled)

Action

None.

Postcondition

```
— if (spdifIn.state == uhSpdifIn_StInvalid)
   — *pStatus == uhSpdifIn_ValStatInvalid
   else if ((spdifIn.state == uhSpdifIn_StNotLocked) ||
         (spdifIn.state == uhSpdifIn_StLocked))
   — *pStatus == uhSpdifIn_ValStatUnknown
   else if (spdifIn.state == uhSpdifIn_StAudioFormatFound)
   — *pStatus == uhSpdifIn_ValStatKnown
   — memcmp(*pData,spdifIn.channel[chNr-
     1].channelStatus,uhSpdifIn_NrChannelStatusBytes)==0
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.4.5.2 uhISpdifInNtf

Qualifiers
— callback

Description
This interface should be implemented by any client that wants to receive notifications generated by the SPDIF-in component. The interface contains one function for each kind of notification. However, for those notifications that are channel related, two notifications (one for each possible channel) are represented by one single function which has a channel number as one of its parameters.

Interface ID
`uuid( B57C5948-8607-46ee-BAC5-CE7CDB6737B3 )`

Execution Constraints
None.

7.1.4.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                    cookie,
    [in] uhSpdifIn_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

7.1.4.5.2.2 OnStateChanged

Signature
```
[async] Void OnStateChanged(
    [in] UInt32            cookie,
    [in] uhSpdifIn_State_t status );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that a transition in the operational state of the SPDIF-in logical component occurred.

Parameters

Table 300

Name	Description
`cookie`	The cookie-value that was passed by the client at subscription time.
`status`	The new status of the SPDIF-in logical component.

Return Values
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.4.5.2.3 OnError

Signature
```
[async] Void OnError(
    [in] UInt32            cookie,
    [in] uhSpdifIn_Error_t error );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that an error occurred when extracting the audio data from the SPDIF signal.

Parameters

Table 301

Name	Description
`cookie`	The cookie-value that was passed by the client at subscription time.
`error`	Describes the type of error that occurred.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.4.5.2.4 OnAudioInfoChanged

Signature
```
[async] Void OnAudioInfoChanged(
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32                          cookie,
    [in] uhSpdifIn_ValStat_t             status,
    [in] uhSpdifIn_ChannelNumber_t       chNr,
    [in] uhSpdifIn_ChannelStatusFormat_t csFormat,
    [in] uhSpdifIn_AudioFormat_t         type,
    [in] uhSpdifIn_Emphasis_t            emphasis,
    [in] uhSpdifIn_Category_t            category,
    [in] UInt32                          samplingFrequency,
    [in] UInt8                           audioSampleWordLength );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that the format of the audio data carried in the SPDIF signal changed. The client must have subscribed on notification uh`SpdifIn_AudioInfoChanged`.

Parameters

Table 302

Name	Description
`cookie`	The cookie-value that was passed by the client at subscription time.
`status`	Indicates the validity of the other parameters of this function.
`chNr`	The channel on which the format change happened.
`csFormat`	Describes the channel status format (either “consumer” or “professional”).
`type`	Describes the audio type (e.g. “PCM”, “MPEG” etc.)
`emphasis`	Describes the kind of pre-emphasis that was applied to the audio data carried by the incoming SPDIF signal.
`category`	Describes the equipment that generated the SPDIF signal.
`samplingFrequency`	Defines the sampling frequency of the digital audio stream in Hz.
`audioSampleWordLength`	Defines the number of bits of each sample of the audio data stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.4.5.2.5 OnChannelInfoChanged

Signature
`[async]` `Void OnChannelInfoChanged(`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32                         cookie,
    [in] uhSpdifIn_ValStat_t            status,
    [in] uhSpdifIn_ChannelNumber_t      chNr,
    [in] UInt8                          channelStatusMode,
    [in] UInt8                          sourceNumber,
    [in] UInt8                          channelNumber );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that the channel information of the specified channel changed.

Parameters

Table 303

Name	Description
`cookie`	The cookie-value that was passed by the client at subscription time.
`status`	Indicates the validity of the other parameters of this function.
`chNr`	The channel on which the format change happened.
`channelStatusMode`	Indicates the channel status mode of the SPDIF signal.
`sourceNumber`	The new source number of this channel [0..15].
`channelNumber`	The new logical channel number of this channel [0..15].

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.4.5.2.6 OnCopyrightInfoChanged

Signature

```
[async] Void OnCopyrightInfoChanged(
    [in] UInt32                       cookie,
    [in] uhSpdifIn_ChannelNumber_t    chNr,
    [in] uhSpdifIn_CopyrightStatus_t  copyrightStatus );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This notification function is used to inform the subscribed client that the copyright status of the audio data changed.

Parameters

Table 304

Name	Description
cookie	The cookie-value that was passed by the client at subscription time.
chNr	The channel for which the copyright status change happened.
copyrightStatus	Describes the new copyright status of the carried audio data of the specified channel.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

7.1.4.5.2.7 OnClockAccuracyChanged

Signature
```
[async] Void OnClockAccuracyChanged(
    [in] UInt32                    cookie,
    [in] uhSpdifIn_ChannelNumber_t chNr,
    [in] uhSpdifIn_ClockAccuracy_t clockAccuracy );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that the clock accuracy of (one of) the monitored channel(s) changed.

Parameters

Table 305

Name	Description
cookie	The cookie-value that was passed by the client at subscription time.
chNr	The channel on which the clock accuracy changed.
clockAccuracy	The new clock accuracy of the SPDIF input signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.1.4.5.2.8 OnChannelStatusChanged

Signature
```
[async] Void OnChannelStatusChanged(
    [in] UInt32                    cookie,
    [in] uhSpdifIn_ValStat_t       status,
    [in] uhSpdifIn_ChannelNumber_t chNr,
    [in] UInt8 pChannelStatus[uhSpdifIn_NrChannelStatusBytes] );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that any bit in the channel status of the specified channel has changed. A reference to the raw (unformatted) channel status bytes is passed. The client is responsible for correct interpretation of these data bytes. Note that this is a low-level access interface to channel status information; most of the channel status data can be retrieved in a formatted form by means of other notifications.

The client must have subscribed on notification uh`SpdifIn_ChannelStatusChanged`.

Parameters

Table 306

Name	Description
`cookie`	The cookie-value that was passed by the client at subscription time.
`status`	Indicates the validity of the other parameters of this function.
`chNr`	The channel on which the channel status change happened.
`pChannelStatus`	Pointer to the raw channel status bytes. The lifetime of the referred data is limited to this function only. The client must make a local copy if he wants to refer to this data after return.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
— If the client intends to use the channel status data even after returning from this call-back, the client has to make a local copy because the lifetime of the passed data is unknown from the client's point of view.

7.1.4.5.2.9 OnUserData

Signature
```
[async] Void OnUserData(
    [in] UInt32                    cookie,
    [in] uhSpdifIn_ValStat_t       status,
    [in] uhSpdifIn_ChannelNumber_t chNr,
    [in, size_is(size)] UInt8*     pUserData,
    [in] UInt32                    size );
```

Qualifiers
None.

Description
This notification function is used to inform the subscribed client that a new packet with user data was extracted from the SPDIF signal. A reference to the raw (unformatted) user data bytes is passed. The client is responsible for correct interpretation of these user data bytes.

The client must have subscribed on notification uh`SpdifIn_UserData`.

Parameters

Table 307

Name	Description
`cookie`	The cookie-value that was passed by the client at subscription time.
`status`	Indicates the validity of the other parameters of this function.
`chNr`	The channel from which the user data was extracted.
`pUserData`	Pointer to the raw user data bytes.
`size`	Size of the user data expressed in number of bytes.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

Remarks
— If the client intends to use the user data even after returning from this call-back, the client has to make a local copy because the lifetime of the passed data is unknown from the client's point of view.

7.1.5 UhlSpdifOut

7.1.5.1 Concepts

7.1.5.1.1 Introduction

In Figure 34 a schematic view of the SPDIF-out logical component is given. As depicted, the SPDIF-out component has one input stream carrying digital audio data with accompanying meta-data. The SPDIF-out component transforms this digital audio stream into an IEC 60958

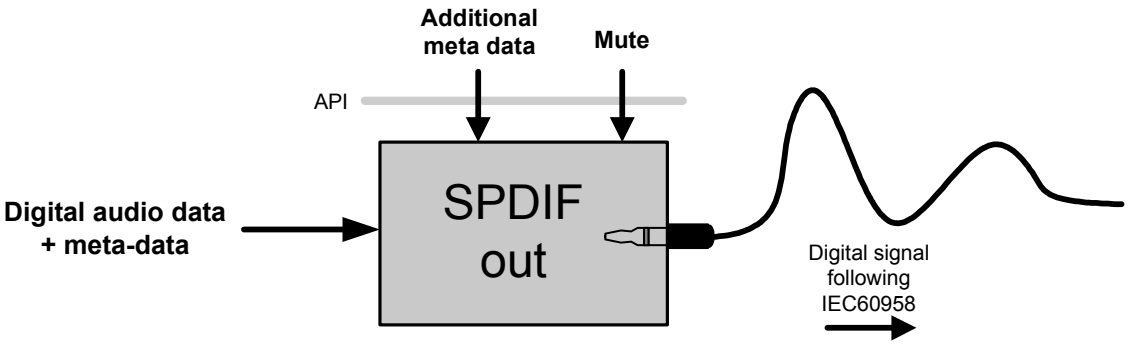


Figure 34 — Data flow through the SPDIF-out logical component.

compliant output signal. The output of the SPDIF-out component is *always* a valid SPDIF signal, even if there is no audio input at all. In these cases, the SPDIF-out component will autonomously *mute* the output signal. The client of SPDIF-out is also able to mute / unmute the output explicitly. The content of a muted SPDIF output signal is elaborated in subclause 7.1.5.1.3. Possible unwanted audible artefacts due to mute/unmute transitions will be suppressed by SPDIF-out in case of linear PCM audio data input.

The output signal contains next to the audio data also some "meta-data". This meta-data is used to describe the properties of the carried audio data. The composition of the meta-data in the output signal depends on the audio input type:

— In case the input audio stream was extracted from a SPDIF source (e.g. SPDIF-in), all meta-data is already present in the input stream and is simply duplicated in the generated SPDIF output signal and the client is not able to alter the meta-data that is inserted in the SPDIF output signal.

— In case the input audio stream does not originate from SPDIF input (e.g. digitized analog audio), not all of the required meta-data to be merged in the generated output signal is present in the input stream. In this case, missing meta-data should be provided by the client of the SPDIF-out logical component. If not provided, default values will be used for inserting meta-data.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.1.2 Meta-data generation

The output SPDIF IEC 60958 compliant signal consists of two multiplexed data streams, called "channel[7]". Each of the two IEC 60958 channels contains next to audio data also meta-data. This meta-data consists of the so-called "channel status data" and "user data". The first describes the properties of the carried audio data, the second is of free-format type and is rarely used for consumer electronic products. The channel status data has a fixed format that is determined by the IEC 60958 standard. To be more precise: the format of the channel status data has two variants: either the "consumer type" or the "professional type".

The user data is not bound to any format. Its use is therefore bound by end-to-end agreements between equipment providers. Unlike the static nature of the channel status data, the user data might be varying for a specific application. The user data could for instance be used to implement an auxiliary data stream.

The way the channel status bytes and user data bytes are generated for the SPDIF output signal depends on the source of the audio data carried by the input stream of the SPDIF-out component:

— If the audio input data originates from a SPDIF input signal, both the user data and channel status data are present in the input stream and are copied from the input stream to the generated output stream unchanged.

— If the audio input contains audio that does not originate from a SPDIF signal (e.g. linear-PCM audio data that was digitized from an analog audio signal), both the user data and channel status bytes must be generated by the SPDIF-out logical component. By default all bytes belonging to the user data are set to "0". This may be overruled if the client explicitly provides user data[8]. The generated channel status data is always of the "consumer type". Most of the channel-status information like sample-rate, and word length of a sample can be extracted from the meta-data of the input stream and can be inserted in the generated SPDIF signal. Other channel status information that cannot be generated automatically must explicitly be provided by the client of the SPDIF-out component. Details about the generation of channel status information are described in Annex E.
The channel status is rather static information and once set, will change very rarely. The same channel status information is repeatedly inserted in the generated SPDIF output signal until the client of the SPDIF-out component changes the channel status information. The user data on the other hand, can best be regarded as a continuous bit-stream. The content of the inserted user data changes continuously. With the interface specified in this document, the client can set a buffer with user data. Unless the client sets a new buffer, the current buffer will be used over and over again. Details about how channel status and user data is inserted in the generated SPDIF signal can be found in [21].

7.1.5.1.3 Muting the output of SPDIF-out

As depicted in Figure 34, the client is able to mute or un-mute the output of the SPDIF-out component explicitly. Besides, the SPDIF-out component may also mute / unmute the output signal autonomously in case the component runs out of audio input. The general mute state is derived from the explicit client mute state and autonomous mute state by a logical OR function. This is depicted in Table 308.

[7] Originally each channel contained audio data for either the right and left component of stereo audio. Yet, a SPDIF signal may contain multiple channels (e.g. for surround sound) or even data belonging to several multiplexed audio sources. The term "channel" therefore does not necessarily reflect a *physical* channel.

[8] Support of user data insertion is a configuration item of the SPDIF-out logical component. Not all implementations of the SPDIF-out component may support user data insertion.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 308 — Resulting mute state due to auto mute versus explicit client mute

	autonomous mute	
client mute	**mute**	**unmute**
mute	mute	mute
unmute	mute	unmute

If the resulting mute state of SPDIF-out changes, a uh`SpdifOut_MuteStateChanged` notification is generated.

If the SPDIF-out component is muted, a special SPDIF mute signal compliant to IEC 60958 is generated. The content of the mute signal depends on the format of the last received audio input data:

— In case of linear PCM audio data, all sample data words of the output signal are set to '0'.

— In case of encoded audio compliant to the IEC60137 standard, all audio burst packets are replaced by so-called "Null" burst packets.

In the former case (linear PCM), the SPDIF-out component guarantees a smooth transition to mute/unmute: audible artefacts are suppressed.

If the SPDIF-out component mutes before it received any audio input data before, a linear PCM audio signal is generated with all data words set to '0'.

7.1.5.1.4 Inserting user data bytes

The SPDIF standard (IEC 60958) supports user data. User data is auxiliary data for which the content is not defined by the SPDIF standard. The meaning of the user data is subject to end-to-end agreements between SPDIF equipment manufacturers.

Whether SPDIF-out supports insertion of user data is subject to configuration.

Though user data can be seen as an auxiliary data stream from which the content changes constantly, the inserted user data controlled by the client of the SPDIF-out component is rather static. That means that the user data is fixed and the same user data is inserted in the generated SPDIF signal repeatedly. In this case, the client of SPDIF-out has to specify user data only once by means of function `SetUserData`. The specified user data will be inserted in the generated output signal repeatedly until the client specifies new user data by invoking `SetUserData` again.

A more dynamic insertion method can also be thought of in which the inserted user data changes continuously. In that case the client has to provide a new buffer with user data before the previous buffer is used. This mechanism is not supported by the interfaces specified in this document.

7.1.5.2 Types & Constants

7.1.5.2.1 Public Types & Constants

7.1.5.2.1.1 Error Codes

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.2.1.2 uhSpdifOut_Notification_t

Signature

```
typedef enum _uhSpdifOut_Notification_t
{
    uhSpdifOut_SubscriptionChanged = 0x00000001,
    uhSpdifOut_DataSourceChanged   = 0x00000002,
    uhSpdifOut_MuteStateChanged    = 0x00000004
} uhSpdifOut_Notification_t, *puhSpdifOut_Notification_t;
```

Qualifiers
Enum-element.

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhISpdifOutNtf`.

Values

Table 309

Name	Description
`uhSpdifOut_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhSpdifOut_DataSourceChanged`	The source of the input stream changed. As a result of this, the availability of channel status data and user data in the input stream changed.
`uhSpdifOut_MuteStateChanged`	The mute state output signal of the SPDIF-out component changed from “muted” to “unmuted” or vice-versa.

7.1.5.2.1.3 uhSpdifOut_Notifications_t

Signature

```
typedef Int32 uhSpdifOut_Notifications_t,
              *puhSpdifOut_Notifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`SpdifOut_Notification_t`. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

7.1.5.2.1.4 uhSpdifOut_AllNotifications

Signature

```
const uhSpdifOut_Notifications_t
      uhSpdifOut_AllNotifications = 0x00000007;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set of all values of type uh`SpdifOut_Notification_t`. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

7.1.5.2.1.5 uhSpdifOut_ChannelNumber_t

Signature
```
typedef enum _uhSpdifOut_ChannelNumber_t {
    uhSpdifOut_Channel1    = 0,
    uhSpdifOut_Channel2    = 1
} uhSpdifOut_ChannelNumber_t, *puhSpdifOut_ChannelNumber_t;
```

Qualifiers
None.

Description
The uh`SpdifOut_ChannelNumber_t` type enumerates the two channels which are carried in the so-called "sub-frames" of a SPDIF signal. See [21] for details about sub-frames.

Values

Table 310

Name	Description
`uhSpdifOut_Channel1`	Specifies the first channel (sub-frame)
`uhSpdifOut_Channel2`	Specifies the second channel (sub-frame)

Remarks
— Note that the channel number does not *necessarily* address the left and right channels of a stereo audio signal. AC-3, for instance, distinguishes 5 audio channels for surround sound.

7.1.5.2.1.6 uhSpdifOut_CopyrightStatus_t

Signature
```
typedef enum _uhSpdifOut_CopyrightStatus_t
{
    uhSpdifOut_CrUnknown     = 0,
    uhSpdifOut_CrNone        = 1,
    uhSpdifOut_CrPreRecorded = 2,
    uhSpdifOut_CrHomeCopy    = 3
} uhSpdifOut_CopyrightStatus_t, *puhSpdifOut_CopyrightStatus_t;
```

Qualifiers
None.

Description
The uh`SpdifOut_CopyrightStatus_t` type is used to describe the copyright status of the audio data in the generated SPDIF signal. This type combines 2 copyright properties into one enumeration: the copyright property (copyrighted or not) and the generation status (number of copies allowed) of the data. If a signal carries copyrighted material, the copyright status is either uh`SpdifOut_CrPreRecorded` or uh`SpdifOut_CrHomeCopy`. If the audio signal is not copyrighted, the copyright state is `uhSpdifOut_CrNone`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 311

Name	Description
uhSpdifOut_CrUnknown	No copyright information present (always the case for the professional format).
uhSpdifOut_CrNone	Copyright information is present. The audio data carried in the SPDIF signal is not copyrighted.
uhSpdifOut_CrPreRecorded	The audio data carried in the SPDIF signal is copyrighted. One more copy is allowed.
uhSpdifOut_CrHomeCopy	The audio data carried in the SPDIF signal is copyrighted. Copying is not allowed.

7.1.5.2.1.7 uhSpdifOut_DataSource_t

Signature

```
typedef enum _uhSpdifOut_DataSource_t
{
    uhSpdifOut_DsSpdif    = 0x00000001,
    uhSpdifOut_DsNonSpdif = 0x00000002
} uhSpdifOut_DataSource_t, *puhSpdifOut_DataSource_t;
```

Qualifiers
None.

Description
The uhSpdifOut_DataSource_t type enumerates the source of the input stream of the SPDIF-out logical component.

Values

Table 312

Name	Description
uhSpdifOut_DsSpdif	The input stream originates from a SPDIF signal. Both, user data and channel status data is available in the input stream.
uhSpdifOut_DsNonSpdif	The input stream does not originate from a SPDIF signal. No user data and only part of the channel status information is present in the input stream.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.2.2 Model Types & Constants

7.1.5.2.2.1 ConsumerChannelStatus_t

Signature

```
typedef struct _ConsumerChannelStatus_t
{
    UInt8                          channelStatusMode;
    UInt8                          sourceNumber;
    UInt8                          channelNumber;
    uhSpdifOut_CopyrightStatus_t copyrightStatus;
} ConsumerChannelStatus_t, *pConsumerChannelStatus_t;
```

Qualifiers
None.

Description
The `ConsumerChannelStatus_t` type represents the portion of the Channel Status bits that must be provided by the client in case the input stream does not originate from a SPDIF signal. The mapping of the data in this structure upon the Channel Status Bits as defined by IEC80958 is described in Annex E.

Fields

Table 313

Name	Description
`channelStatusMode`	This field should always be set to '0'. All other values are reserved for future extension.
`sourceNumber`	Defines the source number for this channel.
`channelNumber`	Defines the channel number for this channel. Note that this is channel number does not address a physical channel of the SPDIF signal but rather a logical channel ID. — 0 means "do not take into account" (see [21]). — 1 indicates, in a stereo configuration, the left channel. — 2 indicates, in a stereo configuration, the right channel.
`copyrightStatus`	Defines whether the audio content is copyrighted and if so, what the generation status of the audio content is (i.e. whether a copy of the audio data is allowed).

Constraints
— `0 <=` sourceNumber `<= 15`
— `0 <=` *channelNumber* `<= 15`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.3 Logical Component

7.1.5.3.1 Interface-Role Model

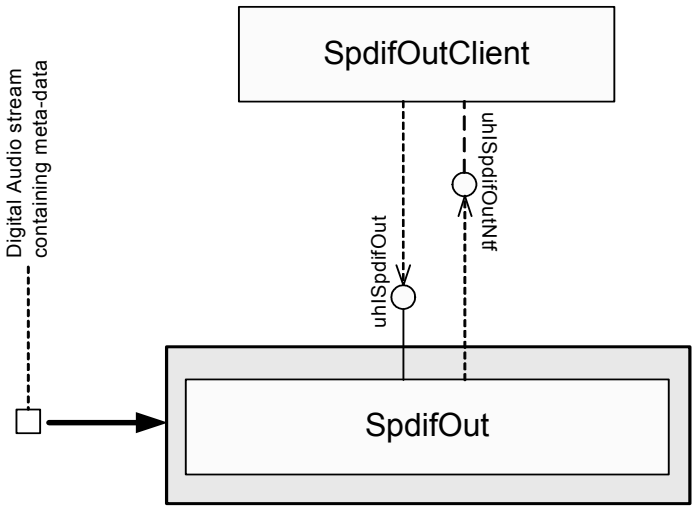


Figure 35 — Interface-Role Model of the SPDIF-out logical component

Figure 35 represents the interface role model for the SPDIF-out logical component. It shows the names and roles that the SPDIF-out logical component and its client have. The role `SpdifOut` provides only one interface to its client: `uhISpdifOut`. The `uhISpdifOut` interface provides the clients control over the SPDIF-out component.

In case events happen in the SPDIF-out component, the `SpdifOut` role might generate notifications. If the client of `SpdifOut` provides an interface of type `uhISpdifOutNtf`, these notifications can be received by the client.

The `SpdifOut` role has only one input stream, no output stream but generates a SPDIF output signal.

7.1.5.3.2 Diversity

7.1.5.3.2.1 Provided Interfaces

Table 314

Role	Interface	Presence
`SpdifOut`	`uhISpdifOut`	mandatory
`SpdifOutClient`	`uhISpdifOutNtf`	optional

7.1.5.3.2.2 Configurable Items

Table 315

Role	Attribute
`SpdifOut`	`Bool userDataSupp`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Not all implementations of `SpdifOut` may support insertion of user data in the generated output SPDIF signal. The `userDataSupp` attribute makes the user data insertion configurable.

7.1.5.3.3 Instantiation

7.1.5.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 316

Type	Object	Multiplicity
`SpdifOut`	`spdifOut`	1

7.1.5.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

- `spdifOut.clientMute = True`
- `spdifOut.autoMute is undefined`
- `spdifOut.dataSource is undefined`
- `spdifOut.uBits = Null`
- `spdifOut.uBitsSize = 0`
- for each i in `[0..1]`:
  - `spdifOut.channel[i].channelStatusMode = 0`
  - `spdifOut.channel[i].sourceNumber = 0`
  - `spdifOut.channel[i].channelNumber = 0`
  - `spdifOut.channel[i].copyrightStatus = uhSpdifOut_CrNone`

7.1.5.3.4 Execution Constraints

The SPDIF-out component acts as an autonomous component. This means that once instantiated, the behavior of the component does not require interaction of its client. The interaction of the client is restricted to:

a) mute or un-mute the output audio signal of the SPDIF-out component.

b) provide channel status information (and possibly user data) that is used in case the input source is “non-SPDIF” audio data.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.4 Roles

7.1.5.4.1 SpdifOut

Signature

```
role SpdifOut
{
    const Bool              userDataSupp;
    Bool                    clientMute;
    Bool                    autoMute;
    uhSpdifOut_DataSource_t dataSource;
    UInt8*                  uBits;
    UInt32                  uBitsSize;
    ConsumerChannelStatus_t channel[2];
}
```

Qualifiers

— root

Description

This role models an instance of a SPDIF-out logical component, providing an interface of type `uhISpdifOut`.

Independent Attributes

Table 317

Name	Description
`userDataSupp`	Describes whether user data insertion is supported by the concerned implementation of SPDIF-out.
`clientMute`	Describes whether the output signal of SPDIF-out is muted by explicit client interaction or not.
`autoMute`	Describes whether the output signal of SPDIF-out is muted as a result of missing input audio data.
`dataSource`	Defines the type of streaming input: `uhSpdifOut_DsSpdif` or uh`SpdifOut_DsNonSpdif`.
`uBits`	Contains the buffer that is inserted in the user-bit fields of the generated SPDIF-signal. This buffer must be provided by the client of the SPDIF component (only if the format of the source data is `uhSpdifOut_DsNonSpdif`, in other cases the user data is copied from the input stream). If the buffer is set to Null, the user data bits in the generated SPDIF signal are set to “0”.
`uBitsSize`	Size of the user data buffer uBits in bytes.
`channel`	Contains the variable part of the channel status information that can be specified by the client. This attribute is only taken into account if `dataSource == uhSpdifOut_DsNonSpdif`.

Invariants

None.

Instantiation

See subclause 7.1.5.3.3 on page 370.

Streaming Behavior

The `SpdifOut` role has only one streaming input, no streaming output but has a SPDIF output *signal*. The input stream contains either audio data that originates from SPDIF-in or from any other digital audio source.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Active Behavior

Table 318

Event	Action
The input source of the SPDIF-out component changed.	For each subscription on notification "uh`SpdifOut_DataSourceChanged`" function `OnDataSourceChanged` is invoked.
The SPDIF-out component runs out of streaming input data or receives streaming input data again after some period of absence of streaming input data.	The generated output is muted/unmuted and for each subscription on notification uh`SpdifOut_MuteStateChanged` function `OnMuteStateChanged` is invoked.

Execution Constraints
None.

7.1.5.4.2 SpdifOutClient

Signature
`role` `spdifOutClient {}`

Qualifiers
— actor

Description
This role models the client of the SPDIF-out logical component.

Execution Constraints
None.

7.1.5.5 Interfaces

7.1.5.5.1 uhISpdifOut

Qualifiers
None.

Description
The `uhISpdifOut` interface is the only interface of a SPDIF-out logical component. This interface is used to control the SPDIF-out logical component.

Interface ID
`uuid( FD47C6C1-6231-42b4-91F5-27BAE795D09B )`

Execution Constraints
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhISpdifOutNtf*            pINotify,
    [in] UInt32                     cookie,
    [in] uhSpdifOut_Notifications_t notifs);
```

Qualifiers

— subscribe-function

7.1.5.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe(
    [in] uhISpdifOutNtf*            pINotify,
    [in] UInt32                     cookie,
    [in] uhSpdifOut_Notifications_t notifs);
```

Qualifiers

— unsubscribe-function

7.1.5.5.1.3 GetUserDataSupp

Signature

```
uhErrorCode_t GetUserDataSupp(
   [out] Bool* pUserDataSupp);
```

Qualifiers

— synchronous

— thread-safe

Description

The client of the SPDIF-out component may use this function to query whether insertion of user data is supported by the concerned implementation or not. If *pUserDataSupp is `False` on function exit, user data insertion is not supported. Note that user data insertion is only possible if the data source of the input stream is not SPDIF. In case user data insertion is not supported and the source of the input signal is not SPDIF, all user data bits in the output signal will be set to ‘0’

Parameters

Table 319

Name	Description
`pUserDataSupp`	Output parameter that on exit defines whether user data insertion is supported by the current implementation of the SPDIF-out component. (`True` means that user data insertion is supported).

Return Values

Standard.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
— `True`.

Action
None.

Postcondition
— **pUserDataSupp* `==` `spdifOut.userDataSupp`

7.1.5.5.1.4 SetUserData

Signature
```
uhErrorCode_t SetUserData(
    [in, size_is(size)] UInt8*      pUserData,
    [in] UInt32                     size );
```

Qualifiers
— synchronous

— single-threaded

Description
Function `SetUserData` specifies a new buffer that will be inserted as user data in the generated SPDIF signal. Note that insertion of user data is only supported by a SPDIF-out instance if `GetUserDataSupp` returns `True`.

The user data that is specified by this function will be inserted repeatedly until new user data is specified. (If the size of the specified buffer is not a multiple of 48, the SPDIF-out component will not align the user data on a IEC 60958 Z-preamble.)

This function has only effect on the generated SPDIF output signal in case the source of the input signal is uh`SpdifOut_DsNonSpdif`.If the source of the input signal is uh`SpdifOut_DsSpdif`, the user data of the input signal is inserted into the output signal and the settings done with this function will not have any effect until the source of the input signal changes to uh`SpdifOut_DsNonSpdif`.

Parameters

Table 320

Name	Description
`pUserData`	Input parameter that contains a reference to the user data that is inserted. Setting argument pUserData to Null specifies that all user data in the generated SPDIF signal is set to “0”.
`size`	Size of the buffer that is returned by parameter pUserData in bytes.

Return Values
Standard.

Precondition
— if `(`pUserData`==Null)`
  — *size*`==0`
— if `(`pUserData`!=Null)`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— *size* >= 48

Action
Modify attributes spdifOut.uBits. and spdifOut.uBitsSize

Postcondition
— spdifOut.uBitsSize == size
— if (size>0)
  — for each i in {0 .. size -1}
    — spdifOut.uBits[i] == pUserData[i]
— else spdifOut.uBits == Null

7.1.5.5.1.5 GetDataSource

Signature
```
uhErrorCode_t GetDataSource(
    [out] uhSpdifOut_DataSource_t* pDataSource );
```

Qualifiers
— synchronous

— thread-safe

Description
This function reports the data source of the input stream. The data source is either uhSpdifOut_DsSpdif or uhSpdifOut_DsNonSpdif.

Parameters

Table 321

Name	Description
pDataSource	Output parameter that on function exit contains the source of the current input stream that is used for output signal generation.

Return Values
Standard

Precondition
True.

Action
None.

Postcondition
— **pDataSource* == spdifOut.dataSource

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.5.1.6 Mute

Signature

```
uhErrorCode_t Mute(
     [in] Bool mute);
```

Qualifiers

— synchronous

— single-threaded

Description

The Mute function is used to mute or un-mute the output of the SPDIF-out logical component. Note that due to missing input signal, the SPDIF-out component may also mute autonomously. Autonomous mute overrules a possible unmute by the client of SPDIF-out.

If the SPDIF-out component is muted, the component will still generate a compliant SPDIF signal. The format of the “mute signal” depends on the former content of the input stream. If the input stream complied to the IEC60137 standard (non PCM encoded audio), all data burst packets from the input are replaced with burst packets of type null data in the generated SPDIF output signal. If the input stream was containing linear PCM, all audio samples are replaced with ‘0’.

The SPDIF-out component takes care of a “smooth” transition while muting or un-muting in case the last received input was linear PCM audio, meaning that audible artefacts like “clicks” do not occur. Smooth transitions will not be guaranteed by SPDIF-out for encoded audio like AC-3, MPEG etc.

Parameters

Table 322

Name	Description
`mute`	Defines whether the output of the SPDIF-out component should be muted or not. (`True` means that the output should be muted, `False` means that the output should contain audio that originates from the input stream.)

Return Values

Standard.

Precondition

`True.`

Action

— Modify the `spdifOut.clientMute` attribute.

— If the mute state (resulting from the explicit client mute and the autonomous mute) of the SPDIF-out component changes, a uh`SpdifOut_MuteStateChanged` notification is generated.

Postcondition

— `spdifOut.clientMute ==` mute

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.5.1.7 GetMuted

Signature

```
uhErrorCode_t GetMuted(
    [out] Bool* pIsMuted);
```

Qualifiers

— synchronous

— thread-safe

Description

The `GetMuted` function returns whether the output of the SPDIF-out component is currently muted or not. Note that due to autonomous mute of the SPDIF-out component, the returned mute state of this function may not reflect the mute state that was set by the client with function Mute.

Parameters

Table 323

Name	Description
`pIsMuted`	Output parameter that on exit reflects the current mute state of the SPDIF-out component. (`True` means that the output is muted, `False` means that the output contains audio that originates from the input stream.)

Return Values

Standard.

Precondition

`True.`

Action

None.

Postcondition

— **pIsMuted* = `spdifOut.clientMute` || `spdifOut.autoMute`

7.1.5.5.1.8 GetChannelInfo

Signature

```
uhErrorCode_t GetChannelInfo(
    [in]  uhSpdifOut_ChannelNumber_t chNr,
    [out] UInt8*                     pChannelStatusMode,
    [out] UInt8*                     pSourceNumber,
    [out] UInt8*                     pChannelNumber);
```

Qualifiers

— synchronous

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
The `GetChannelInfo` function returns settings that are used to generate the Channel Status information in case the audio source is uh`SpdifOut_DsNonSpdif`. Because the Channel Status information of channel 1 and 2 may deviate from each other, parameter chNr selects one of the two channels.

Parameters

Table 324

Name	Description
`chNr`	Input parameter that specifies for which channel the Channel Info is requested.
`pChannelStatusMode`	Output parameter that indicates the Channel Status Mode. The returned value should always be “0” according to the IEC 60958 standard.
`pSourceNumber`	Output parameter that on function exit describes the source number of the channel.
`pChannelNumber`	Output parameter that on function exit indicates the channel number of the selected channel. Note that this is the logical channel number rather than the *physical* channel number.

Return Values
Standard

Precondition
`True`

Action
None.

Postcondition
— **pChannelStatusMode* `==` `spdifOut.channel[chNr].channelStatusMode`
— *pSourceNumber `==` `spdifOut.channel[chNr].sourceNumber`
— **pChannelNumber* `==` `spdifOut.channel[chNr].channelNumber`

Remarks
— The `GetChannelInfo` function cannot be used to retrieve the Channel Status Info of the *input* stream in case the input source was SPDIF. It can only be used to retrieve the channel related properties that will be inserted in the generated SPDIF output signal in case the source of the input stream is not SPDIF.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.5.1.9 SetChannelInfo

Signature

```
uhErrorCode_t SetChannelInfo(
    [in] uhSpdifOut_ChannelNumber_t chNr,
    [in] UInt8                      channelStatusMode,
    [in] UInt8                      sourceNumber,
    [in] UInt8                      channelNumber);
```

Qualifiers

— synchronous

— single-threaded

Description

The `SetChannelInfo` function specifies part of Channel Status bits that will be inserted in the meta-data of `channel[chNr]` of the generated SPDIF output signal. This is only applicable if the source of the input stream is uh`SpdifOut_DsNonSpdif`. If the source of the input signal is uh`SpdifOut_DsSpdif`, the Channel Status bits of the input signal will be copied into the output signal and the settings done with this function will not have any effect until the source changes to uh`SpdifOut_DsNonSpdif`.

Parameters

Table 325

Name	Description
`chNr`	Specifies for which of the two channels the channel status information must be used.
`channelStatusMode`	Parameter that indicates the Channel Status Mode. According to the IEC 60958 standard, this parameter should always be set to “0”. All other values are reserved for future use.
`sourceNumber`	Specifies the source number of the audio signal of the specified channel.
`channelNumber`	Specifies the logical channel number of the audio signal of the specified channel.

Return Values

Standard.

Precondition

— `0 <= sourceNumber <= 15`

— `0 <= channelNumber <= 15`

Action

— Modify attributes `spdifOut.channel[chNr].`channelStatusMode, `spdifOut.channel[chNr].`*sourceNumber* and `spdifOut.channel[chNr].`channelNumber

Postcondition

— `spdifOut.channel[chNr].`*channelStatusMode* == *channelStatusMode*

— `spdifOut.channel[chNr].`sourceNumber == sourceNumber

— `spdifOut.channel[chNr].`channelNumber == channelNumber

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.5.1.10 GetCopyrightInfo

Signature

```
uhErrorCode_t GetCopyrightInfo(
    [in]  uhSpdifOut_ChannelNumber_t    chNr,
    [out] uhSpdifOut_CopyrightStatus_t* pCopyrightStatus);
```

Qualifiers

— synchronous

— thread-safe

Description

The `GetCopyrightInfo` function returns the copyright related properties that are inserted in the channel status bytes of `channel[chNr]` if the source of the input stream is not SPDIF. The copyright information contains two properties w.r.t. copyright:

— whether the data in the input stream is copyrighted;

— the generation state (whether the user is allowed to make one more copy of the audio data).

The two properties are combined into one parameter *pCopyrightStatus*.

Parameters

Table 326

Name	Description
`chNr`	Specifies for which of the two channels the copyright properties must be returned.
`pCopyrightStatus`	Output parameter that on exit will indicate the copyright information w.r.t. the audio data carried in the channel specified with chNr.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— **pCopyrightStatus* == `spdifOut.channel[`chNr`].copyrightStatus`

Remarks

— `GetCopyrightInfo`() cannot be used to retrieve the Copyright info of the *input* stream in case the input source is uh`SpdifOut_DsSpdif`. It can only be used to retrieve the copyright related properties that will be inserted in the generated SPDIF output signal in case the source of the input stream is (or becomes) uh`SpdifOut_DsNonSpdif`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.1.5.5.1.11 SetCopyrightInfo

Signature

```
uhErrorCode_t SetCopyrightInfo(
    [in] uhSpdifOut_ChannelNumber_t   chNr,
    [in] uhSpdifOut_CopyrightStatus_t copyrightStatus);
```

Qualifiers

— synchronous

— single-threaded

Description

The `SetCopyrightInfo` function specifies the copyright related part of the Channel Status information that will be inserted in the generated SPDIF compliant output signal in case the source of the input stream is uh`SpdifOut_DsNonSpdif`. If the source of the input signal is uh`SpdifOut_DsSpdif`, the copyright properties of the input signal are copied into the output signal and the settings done with this function will not have any effect until the source of the input audio stream becomes uh`SpdifOut_DsNonSpdif`.

The copyright information contains two properties w.r.t. copyright:

— whether the data in the stream is copyrighted;

— the generation state (whether the user is allowed to make one more copy of the audio data).

Parameters

Table 327

Name	Description
`chNr`	Specifies for which of the two channels the copyright properties must be inserted.
`copyrightStatus`	Specifies the copyright related properties.

Return Values

Standard

Precondition

— *copyrightStatus* `!= uhSpdifOut_CrUnknown`

Action

— Modify `spdifOut.channel[`chNr`].copyrightStatus`.

Postcondition

— `spdifOut.channel[`*chNr*`].copyrightStatus ==` copyrightStatus

7.1.5.5.2 uhISpdifOutNtf

Qualifiers

— callback

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The `uhISpdifOutNtf` interface describes an interface that should be implemented by clients of the SPDIF-out component in order to receive notifications that are generated by SPDIF-out.

Interface ID
`uuid( D1222035-0FFF-4222-923F-DC283BD101F8 )`

Execution Constraints
None.

7.1.5.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                     cookie,
    [in] uhSpdifOut_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

7.1.5.5.2.2 OnDataSourceChanged

Signature
```
[async] Void OnDataSourceChanged(
    [in] UInt32                  cookie,
    [in] uhSpdifOut_DataSource_t dataSource );
```

Qualifiers
None.

Description
In case the source of the streaming input data of SPDIF-out changes, this function is invoked to inform the client(s) about this event. The new source of the input is passed by means of parameter *dataSource*.

Parameters

Table 328

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`dataSource`	Specifies the new source of the streaming input.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
Any call-back compliant action.

Postcondition
`True`

7.1.5.5.2.3 OnMuteStateChanged

Signature

```
[async] Void OnMuteStateChanged(
    [in] UInt32 cookie,
    [in] Bool   isMuted );
```

Qualifiers
None.

Description
This function is invoked if the mute state of the SPDIF-out component changes from “muted” to “unmuted” or vice-versa. The reason for this change can be explicit interaction by the client with use of function Mute or autonomous mute/unmute by the SPDIF-out component.

Parameters

Table 329

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`isMuted`	Indicates the mute state of the SPDIF-out component (`True`=muted, `False`=not muted).

Return Values
None.

Precondition
`True`

Action
Any notification compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2 Audio processing and rendering

7.2.1 UhlAavl

7.2.1.1 Concepts

7.2.1.1.1 Automatic Volume Leveling

Audio signals from different sound sources do not necessarily have the same volume level, or the volume level may vary with the audio content e.g. the volume in advertisements is usually louder than in other programs like a movie. This is often experienced as disturbing by the end-user. Automatic Volume Leveling (AVL) is a feature that compensates for this volume changes, and results in less annoyance for the end-user.

AVL uses measurements of the input volume level, possibly measured in upstream logical components (in the case of decoded multi-channel audio input streams). Because of the measurements and possible delays in the AVL operation, the response time of AVL is not instantaneous and can take a short time.

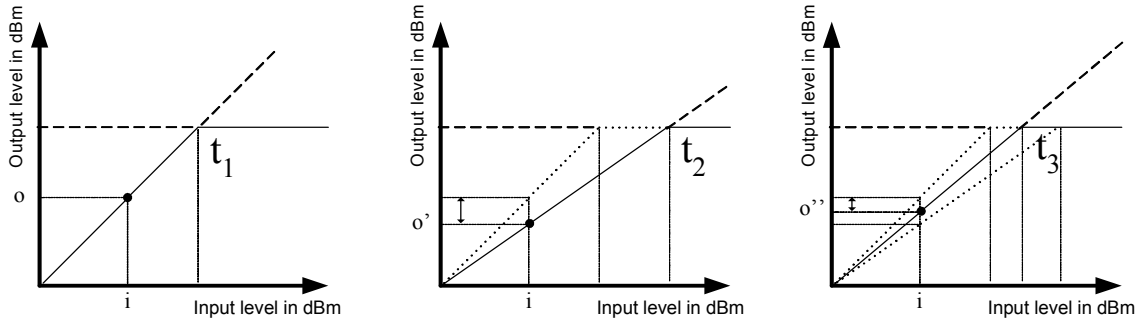


Figure 36 — Example of the Automatic Volume Leveling mechanism.

In Figure 36, the AVL mechanism is depicted:

— At time t_1, the average input volume level is below or equal to the maximum volume level, the volume level of o at the output is equal to the volume level i at the input;

— At time t_2, the average input volume level has exceeded the maximum volume level, the volume level of o' at the output is reduced compared to the volume level i at the input, due to the gain adjustment of the AVL feature;

— At time t_3, the average input volume level is still above the maximum volume level, the volume level of o'' at the output is still adjusted - but less - compared to the volume level i at the input, due to the gain adjustment of the AVL feature;

— The moment the average input volume level is again below or equal to the maximum volume level, the volume level of the output is equal to the volume level at the input, as depicted at time t_1.

The client can enable, disable, and set different levels of AVL. A higher level of AVL results in a more limited range of the output volume and/or different response times, depending on the platform realization.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

AVL may be autonomously de-activated in certain cases where its application conflicts with other features in the sound signal path. Changes in the actual enabling status of AVL will be notified.

7.2.1.2 Types & Constants

7.2.1.2.1 Public Types & Constants

7.2.1.2.1.1 Error Codes

None.

7.2.1.2.1.2 uhAavl_ValStat_t

Signature

```
typedef enum _uhAavl_ValStat_t {
    uhAavl_ValStatInvalid = -2,
    uhAavl_ValStatUnknown = -1,
    uhAavl_ValStatKnown   =  1
} uhAavl_ValStat_t, *puhAavl_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 330

Name	Description
uhAavl_ValStatInvalid	The property/properties are temporarily not available.
uhAavl_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhAavl_ValStatKnown	The property/properties are available and determined.

7.2.1.2.1.3 uhAavl_Ntf_t

Signature

```
typedef enum _uhAavl_Ntf_t {
    uhAavl_SubscriptionChanged = 0x00000001,
    uhAavl_ActEnabledChanged   = 0x00000002
} uhAavl_Ntf_t, *puhAavl_Ntf_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAavlNtf`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 331

Name	Description
uhAavl_SubscriptionChanged	Represents OnSubscriptionChanged.
uhAavl_ActEnabledChanged	Represents OnActEnabledChanged.

7.2.1.2.1.4 uhAavl_NtfSet_t

Signature
```
typedef Int32 uhAavl_NtfSet_t, *puhAavl_NtfSet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type

uhAavl_Ntf_t. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

7.2.1.2.1.5 uhAavl_AllNtfs

Signature
```
const uhAavl_NtfSet_t uhAavl_AllNtfs = 0x00000003;
```

Qualifiers
— None.

Description
Defines the set of all values of type

uhAavl_Ntf_t. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

7.2.1.2.2 Model Types & Constants

None.

7.2.1.3 Logical Component

7.2.1.3.1 Interface-Role Model

The interface role-model is depicted in Figure 37.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

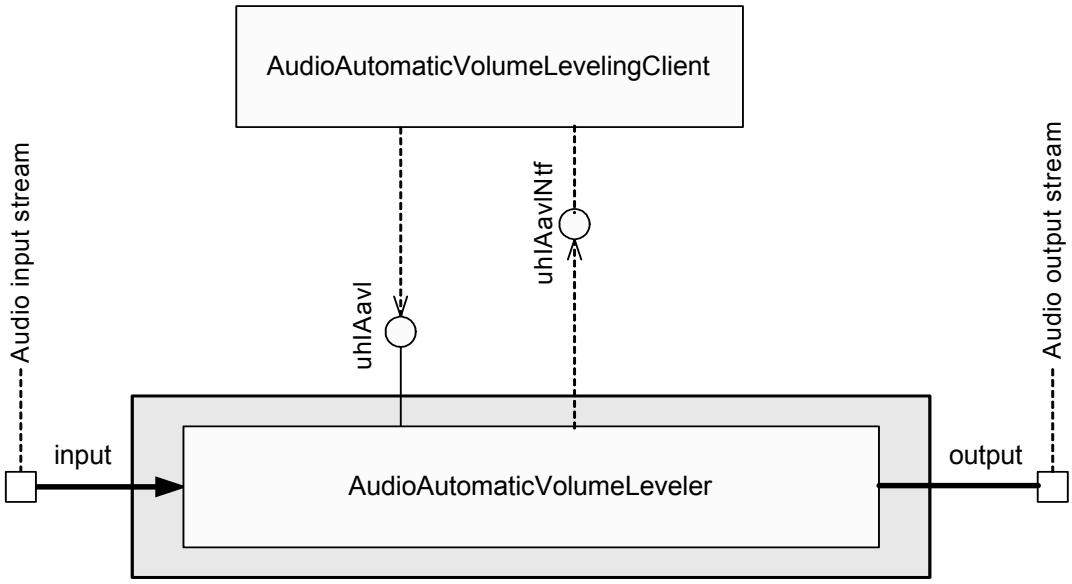


Figure 37 — Interface-Role Model of the Audio Automatic Volume Leveler logical component

The Audio Automatic Volume Leveler logical component (AAVL, grey box) consists of the `AudioAutomaticVolumeLeveler` role. This role provides an interface (`uhIAavl`) for controlling AVL. The control interface is meant for use by the client (`AudioAutomaticVolumeLevelingClient`). This client can optionally provide a call-back interface (`uhIAavlNtf`) allowing it to receive notifications of events in the Audio Automatic Volume Leveler logical component.

7.2.1.3.2 Diversity

7.2.1.3.2.1 Provided Interfaces

Table 332

Role	Interface	Presence
`AudioAutomaticVolumeLeveler`	`uhIAavl`	mandatory
`AudioAutomaticVolumeLevelingClient`	`uhIAavlNtf`	optional

7.2.1.3.2.2 Configurable Items

Table 333

Role	Attribute
`AudioAutomaticVolumeLeveler`	`minLevel`
	`maxLevel`

7.2.1.3.2.3 Constraints

```
minLevel <= maxLevel
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.1.3.3 Instantiation

7.2.1.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 334

Type	Object	Multiplicity
AudioAutomaticVolumeLeveler	aavl	1

7.2.1.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— aavl.setEnabled == False
— aavl.actualEnabledStatus == uhAavl_ValStatInvalid
— aavl.actualEnabled == False
— aavl.level == minLevel

7.2.1.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

7.2.1.4 Roles

7.2.1.4.1 AudioAutomaticVolumeLeveler

Signature

```
role AudioAutomaticVolumeLeveler {
    Bool                setEnabled;
    uhAavl_ValStat_t    actualEnabledStatus;
    Bool                actualEnabled;
    const UInt32        minLevel;
    const UInt32        maxLevel;
    UInt32              level;
}
```

Qualifiers

— root

Description

The AudioAutomaticVolumeLeveler role controls the AVL level. The AVL level can be set and queried and AVL processing can be enabled or disabled.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 335

Name	Description
setEnabled	Indicates the preferred AVL enabling status, True if enabled and False if disabled.
actualEnabledStatus	Validity state of the AVL enabling status in actualEnabled.
actualEnabled	Indicates the actual AVL enabling status, True if enabled and False if disabled.
minLevel	Minimum value for the AVL level.
maxLevel	Maximum value for the AVL level.
level	Current value of the AVL level.

Dependent Attributes
None.

Invariants

— minLevel <= level <= maxLevel

Instantiation
See subclause 7.2.1.3.3.

Streaming Behavior
The AudioAutomaticVolumeLeveler has an audio signal as an input.

When the AVL processing is enabled (aavl.setEnabled is True)

and AVL processing is not de-activated autonomously by the platform, then aavl.actualEnabled is True and the AudioAutomaticVolumeLeveler will perform AVL processing on the audio input stream, using the AVL level set in aavl.level.

This processed stream is passed to the output.

The AVL processing can be disabled as a result of:

— Explicitly disabling by the client (aavl.setEnabled is False)

and/or

— Autonomous de-activation by the platform.

When the AVL processing is disabled (aavl.actualEnabled is False), the stream at the input is passed to the output without any processing.

Active Behavior
In addition to the standard notification behavior (see uhIAavlNtf), AudioAutomaticVolumeLeveler has the following event-driven notification behavior:

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 336

Event	Action
The applicability of AVL towards the audio input stream of the AAVL logical component, has changed.	`Notify OnActEnabledChanged if the enabling status of AVL changes.`

Execution Constraints
None.

7.2.1.4.2 AudioAutomaticVolumeLevelingClient

Signature
`role` `AudioAutomaticVolumeLevelingClient {}`

Qualifiers
— actor

Description
An `AudioAutomaticVolumeLevelingClient` represents a user of the `uhIAavl` interface.

Execution Constraints
— None

7.2.1.5 Interfaces

7.2.1.5.1 uhIAavl

Qualifiers
None.

Description
This interface provides control on the activation/deactivation of the AVL processing and on the level of AVL that is applied to the audio input signal.

Interface ID
`uuid( 5A9F23B0-E331-4025-B024-70684F9C11BA )`

Remarks
— In the following function specifications, the name `aavl` will be used to refer to the provider of this interface; see subclause 7.2.1.3.3.1.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.1.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIAavlNtf       *pINotify,
    [in] UInt32           cookie,
    [in] uhAavl_NtfSet_t  notifs );
```

Qualifiers

— subscribe-function

7.2.1.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIAavlNtf       *pINotify,
    [in] UInt32           cookie,
    [in] uhAavl_NtfSet_t  notifs );
```

Qualifiers

— unsubscribe-function

7.2.1.5.1.3 Enable

Signature

```
uhErrorCode_t Enable ( [in] Bool enable );
```

Qualifiers

— synchronous

— single-threaded

Description

Sets the preferred AVL enabling status.

Parameters

Table 337

Name	Description
`enable`	The preferred AVL enabling status, being `True` if enabled and `False` if disabled.

Return Values

Standard.

Precondition

`True`

Action

— Modify `aavl.setEnabled`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `aavl.setEnabled == enable`

7.2.1.5.1.4 GetEnabled

Signature
```
uhErrorCode_t GetEnabled ( [out] pBool pEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the preferred AVL enabling status.

Parameters

Table 338

Name	Description
`pEnabled`	Output parameter returning the preferred AVL enabling status, being `True` if enabled and `False` if disabled.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pEnabled == aavl.setEnabled`

7.2.1.5.1.5 GetActualEnabled

Signature
```
uhErrorCode_t GetActualEnabled (
    [out] puhAavl_ValStat_t pStatus,
    [out] pBool             pActualEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the actual AVL enabling status.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 339

Name	Description
`pStatus`	Output parameter that indicates the validity of the actual AVL enabling status in the `pActualEnabled` parameter.
`pActualEnabled`	Output parameter returning the actual AVL enabling status, being `True` if enabled and `False` if disabled.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
```
*pStatus == aavl.actualEnabledStatus

*pActualEnabled == aavl.actualEnabled
```

7.2.1.5.1.6 SetLevel

Signature
```
uhErrorCode_t SetLevel ( [in] UInt32 level );
```

Qualifiers
- synchronous
- single-threaded

Description
Sets the AVL level.

Parameters

Table 340

Name	Description
`level`	The AVL level to be set.

Return Values
Standard.

Precondition
- `aavl.minLevel <= level <= aavl.maxLevel`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `aavl.level`

Postcondition
— `aavl.level ==` level

7.2.1.5.1.7 GetLevel

Signature
```
uhErrorCode_t GetLevel ( [out] pUInt32 pLevel );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the AVL level.

Parameters

Table 341

Name	Description
`pLevel`	Output parameter returning the AVL level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*`*pLevel* `== aavl.level`

7.2.1.5.1.8 GetLevelRange

Signature
```
uhErrorCode_t GetLevelRange (
    [out] pUInt32 pMinLevel,
    [out] pUInt32 pMaxLevel );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the minimum and the maximum value for the AVL level.

Parameters

Table 342

Name	Description
`pMinLevel`	Output parameter returning the minimum value for the AVL level.
`pMaxLevel`	Output parameter returning the maximum value for the AVL level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— *pMinLevel `== aavl.minLevel`
— **pMaxLevel* `== aavl.maxLevel`

7.2.1.5.2 uhIAavlNtf

Qualifiers
— callback

Description
This is the notification interface to be provided by clients of the AAVL logical component that wish to receive notifications of events of the AAVL logical component.

Interface ID
`uuid( E2798E9B-5AF0-412C-8E33-51BFCAEAC5D8 )`

7.2.1.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32           cookie,
    [in] uhAavl_NtfSet_t  notifs );
```

Qualifiers
— onsubscriptionchanged-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.1.5.2.2 OnActEnabledChanged

Signature

```
[async] Void OnActEnabledChanged (
    [in] UInt32           cookie,
    [in] uhAavl_ValStat_t status,
    [in] Bool             enable);
```

Qualifiers
None.

Description
This function is called by the `AudioAutomaticVolumeLeveler` whenever it detects a change in the actual enabling status of AVL for the current audio input stream.

Parameters

Table 343

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	Parameter that indicates the validity of the AVL enabling status in the `enable` parameter.
`enable`	The new AVL enabling status. It is `True` if AVL is enabled for the current audio input stream and `False` if AVL is disabled.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.2.2 UhlAbassEnh

7.2.2.1 Concepts

7.2.2.1.1 Introduction

Bass Enhancement (BE) refers to a set of signal processing algorithms that are used to provide improved listening experience for low-frequency (bass) sounds. It is typically applied in the audio frequency range below 120 Hz and its purpose may depend on the kind of bass rendering capabilities of the speaker system.

The bass enhancement algorithms mainly fall into two broad categories:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

a) Those that compensate for some inadequacies in the speakers. This usually pertains to speaker systems that have reduced acoustic outputs at low-frequencies. The effect could take shape of the following:

   1) A bass-booster that compensates for the low output levels of a given speaker system at low frequencies. This is used where the speakers have a reduced, but functional low-frequency response. Often an implementation of this takes the form of compensating the speakers' amplitude (and sometimes phase) response through an "inverse" filter. Frequently speaker compensation systems extend beyond the low-frequency range of frequencies, and cease to be "bass enhancement" algorithms in a pure sense.

   2) A phantom-bass generator that introduces harmonics of the fundamental in the output signal to create the perception of bass. This is usually deployed where the speakers have a severely impaired low-frequency response.

b) Those that regulate subjective sound level in the low-frequency range of frequencies. A typical implementation of this corresponds to a bass-boost filter whose gain reduces with the signal level.

Some of the popularly available Bass Enhancement algorithms are listed as examples in Appendix A.

7.2.2.1.2 Operational Bass Enhancement Modes

The Audio Bass Enhancer logical component has the form of a filter with a single streaming input pin and a single streaming output pin.

The Audio Bass Enhancer logical component supports only generic bass-enhancement settings. The control for the logical component is open loop in the sense that there are no notifications that the client can use.

The Audio Bass Enhancer logical component can support the following bass enhancement modes:

— Bass Compensation (**BC**): Generic name associated with type 1a in subclause 7.2.2.1.1.

— Phantom Bass (**PB**): Generic name associated with type 1b in subclause 7.2.2.1.1.

— Dynamic Bass Enhancement (**DBE**): Generic name associated with type 2 in subclause 7.2.2.1.1.

All modes can be used in conjunction. Each mode has an associated "effect-strength" parameter, which can be set by the user.

7.2.2.2 Types & Constants

7.2.2.2.1 Public Types & Constants

7.2.2.2.1.1 Error Codes

None.

7.2.2.2.1.2 uhAbassEnh_Type_t

Signature

```
typedef enum _uhAbassEnh_Type_t {
    uhAbassEnh_BassCompensation       = 0x00000001,
    uhAbassEnh_PhantomBass            = 0x00000002,
    uhAbassEnh_DynamicBassEnhancement = 0x00000004
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
} uhAbassEnh_Type_t, *puhAbassEnh_Type_t;
```

Qualifiers
— enum-element

Description
Values of this type represent the different types of bass enhancement algorithm as described in subclause 7.2.2.1.2.

Values

Table 344

Name	Description
`uhAbassEnh_BassCompensation`	Corresponds to bass compensation (BC).
`uhAbassEnh_PhantomBass`	Corresponds to phantom bass (PB).
`uhAbassEnh_DynamicBassEnhancement`	Corresponds to dynamic bass enhancement (DBE).

7.2.2.2.1.3 uhAbassEnh_TypeSet_t

Signature
`typedef Int32`

uhAbassEnh_TypeSet_t, `*p`

uhAbassEnh_TypeSet_t;

Qualifiers
— enum-set

Description
Values of this type represent sets of bass enhancement algorithm types that are typically constructed by logical OR-ing of values of type `uhAbassEnh_Type_t`.

Constraints
None.

7.2.2.2.2 Model Types & Constants

7.2.2.2.2.1 NoBassEnhType

Signature
```
const Int32 NoBassEnhType = 0x00000000;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the empty set of bass enhancement algorithm types.

7.2.2.3 Logical Component

7.2.2.3.1 Interface-Role Model

Figure 38 depicts the interface-role model of the Audio Bass Enhancer logical component. It shows the interfaces and the roles involved with this component.

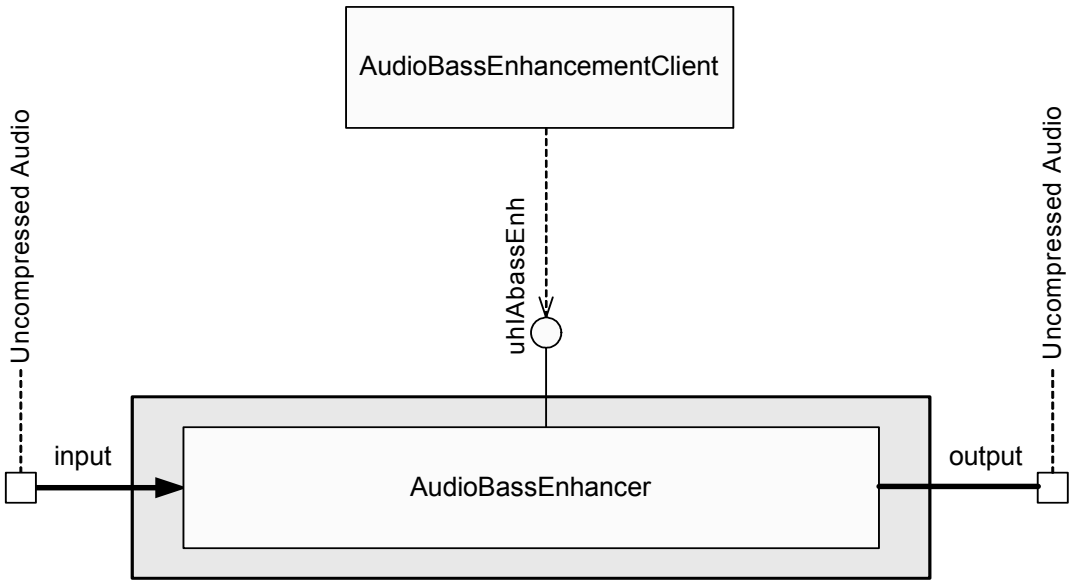


Figure 38 — Interface Role Model

The grey box in Figure 38 denotes an **Audio Bass Enhancer** logical component containing a single `AudioBassEnhancer` role. This role provides one mandatory (solid line) control interface called `UhIAbassEnh` to its client, the `AudioBassEnhancementClient`. There are no notification interfaces since the control is an open-loop one.

7.2.2.3.2 Diversity

7.2.2.3.2.1 Provided Interfaces

Table 345

Role	Interface	Presence
`AudioBassEnhancer`	`UhIAbassEnh`	mandatory

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.3.2.2 Configurable Items

Table 346

Role	Attribute
AudioBassEnhancer	suppBassEnhTypes
	minEffectStrengthDbe
	maxEffectStrengthDbe
	minEffectStrengthPb
	maxEffectStrengthPb
	minEffectStrengthBc
	maxEffectStrengthBc

7.2.2.3.2.3 Constraints

— suppBassEnhTypes != NoBassEnhType
— minEffectStrengthDbe ≤ maxEffectStrengthDbe
— minEffectStrengthPb ≤ maxEffectStrengthPb
— minEffectStrengthBc ≤ maxEffectStrengthBc

7.2.2.3.3 Instantiation

7.2.2.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 347

Type	Object	Multiplicity
AudioBassEnhancer	abe	1

7.2.2.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— abe.currentBassEnhTypes == NoBassEnhType
— abe.effectStrengthBc == minEffectStrengthBc
— abe.effectStrengthPb == minEffectStrengthPb
— abe.effectStrengthDbe == minEffectStrengthDbe

7.2.2.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier “thread-safe”.

7.2.2.4 Roles

AudioBassEnhancer

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Signature

```
role AudioBassEnhancer {
    const
uhAbassEnh_TypeSet_t suppBassEnhTypes;

uhAbassEnh_TypeSet_t       currentBassEnhTypes;
    const Int32                  minEffectStrengthBc;
    const Int32                  maxEffectStrengthBc;
    const Int32                  minEffectStrengthPb;
    const Int32                  maxEffectStrengthPb;
    const Int32                  minEffectStrengthDbe;
    const Int32                  maxEffectStrengthDbe;
    Int32                        effectStrengthBc;
    Int32                        effectStrengthPb;
    Int32                        effectStrengthDbe;
}
```

Qualifiers

— root

Description

The `AudioBassEnhancer` role performs the function of accepting bass enhancement settings from the user, and retrieving these settings on request.

For each type of bass enhancement, an “effect-strength” can be set and queried. The “effect-strength” limit parameters (min/max) providing the range over which each type of bass enhancement may be set by the user, can also be queried.

Independent Attributes

Table 348

Name	Description
`suppBassEnhTypes`	The set of supported bass enhancement types.
`currentBassEnhTypes`	The set of bass enhancement types that are currently activated. A bass enhancement type, once activated by the user, will remain activated, until the user deactivates it. It will not be affected by settings of other bass enhancement types.
`minEffectStrengthBc`	Minimum effect-strength that can be used for bass enhancement type BC.
`maxEffectStrengthBc`	Maximum effect-strength that can be used for bass enhancement type BC.
`minEffectStrengthPb`	Minimum effect-strength that can be used for bass enhancement type PB.
`maxEffectStrengthPb`	Maximum effect-strength that can be used for bass enhancement type PB.
`minEffectStrengthDbe`	Minimum effect-strength that can be used for bass enhancement type DBE.
`maxEffectStrengthDbe`	Maximum effect-strength that can be used for bass enhancement type DBE.
`effectStrengthBc`	Effect-strength used for BC, when bass enhancement type BC is activated.
`effectStrengthPb`	Effect-strength used for PB, when bass enhancement type PB is activated.
`effectStrengthDbe`	Effect-strength used for DBE, when bass enhancement type DBE is activated.

Dependent Attributes

None.

Invariants

— `(currentBassEnhTypes & suppBassEnhTypes) == currentBassEnhTypes`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— `minEffectStrengthBc ≤ effectStrengthBc ≤ maxEffectStrengthBc`
— `minEffectStrengthPb ≤ effectStrengthPb ≤ maxEffectStrengthPb`
— `minEffectStrengthDbe ≤ effectStrengthDbe ≤ maxEffectStrengthDbe`

Instantiation
An `AudioBassEnhancer` is always instantiated as part of an Audio Bass Enhancer; see subclause 7.2.2.3.3.

Streaming Behavior
The `AudioBassEnhancer` role has one audio input stream and one audio output stream. The `AudioBassEnhancer` processes the audio signal of the input stream with the algorithm(s) set in `currentBassEnhTypes` and streams the resulting audio signal out.

Active Behavior
None.

Execution Constraints
None.

AudioBassEnhancementClient

Signature
`role AudioBassEnhancementClient {}`

Qualifiers
— actor

Description
An Audio Bass Enhancement Client represents a user of the `UhIAbassEnh` interface.

Execution Constraints
None.

7.2.2.5 Interfaces

7.2.2.5.1 uhIAbassEnh

Qualifiers
None.

Description
This interface of an `AudioBassEnhancer` provides functions to set and retrieve bass enhancement preferences.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
uuid(6E60D4E3-1E67-4C0C-98DF-CC2A7FE6D73E)

7.2.2.5.1.1 GetSuppBassEnhTypes

Signature
```
uhErrorCode_t GetSuppBassEnhTypes(
    [out] p

uhAbassEnh_TypeSet_t pSuppBassEnhTypes );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns the set of supported bass enhancement types.

Parameters

Table 349

Name	Description
pSuppBassEnhTypes	Output parameter returning the set of supported bass enhancement types.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pSuppBassEnhTypes == abe.suppBassEnhTypes

7.2.2.5.1.2 GetEffectStrengthRange

Signature
```
uhErrorCode_t GetEffectStrengthRange(
    [in] uhAbassEnh_Type_t bassEnhType,
    [out] pInt32 pMinEffectStrength,
    [out] pInt32 pMaxEffectStrength );
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers
— synchronous

— thread-safe

Description
This function returns the range over which the “effect-strength” can be set for a given bass enhancement type.

Parameters

Table 350

Name	Description
`bassEnhType`	The bass enhancement type for which the minimum and maximum the “effect-strength” are queried.
`pMinEffectStrength`	Output parameter returning the minimum “effect-strength” for the bass enhancement type `bassEnhType`.
`pMaxEffectStrength`	Output parameter returning the maximum “effect-strength” for the bass enhancement type `bassEnhType`.

Return Values
— Standard

Precondition
`(bassEnhType & abe.suppBassEnhTypes) != 0`

Action
None.

Postcondition
— If `(bassEnhType == uhAbassEnh_BassCompensation)`
  — `*pMinEffectStrength == abe.minEffectStrengthBc`
  — `*pMaxEffectStrength == abe.maxEffectStrengthBc`
— If `(bassEnhType == uhAbassEnh_PhantomBass)`
  — `*pMinEffectStrength == abe.minEffectStrengthPb`
  — `*pMaxEffectStrength == abe.maxEffectStrengthPb`
— If `(bassEnhType == uhAbassEnh_DynamicBassEnhancement)`
  — `*pMinEffectStrength == abe.minEffectStrengthDbe`
  — `*pMaxEffectStrength == abe.maxEffectStrengthDbe`

7.2.2.5.1.3 SetBassEnh

Signature
```
uhErrorCode_t SetBassEnh(
    [in] uhAbassEnh_Type_t bassEnhType,
    [in] Int32 effectStrength,
    [in] Bool enable);
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers
- synchronous
- single-threaded

Description
This function sets the “effect-strength” and activation status for a given bass enhancement type.

Parameters

Table 351

Name	Description
`bassEnhType`	The type of bass enhancement for which activation and “effect-strength” are set.
`effectStrength`	The “effect-strength” that should be applied for the bass enhancement type `bassEnhType` when this bass enhancement type is activated. It is specified in integral steps within the range corresponding to `bassEnhType`.
`enable`	The activation state for the bass enhancement type `bassEnhType`. It is `True` when the specified bass enhancement type should be active. It is `False` when the specified bass enhancement type should not be active.

Return Values
Standard.

Precondition
- `bassEnhType & abe.suppBassEnhTypes != 0`
- If `(bassEnhType == uhAbassEnh_BassCompensation)`
  - `abe.minEffectStrengthBc <= effectStrength <= abe.maxEffectStrengthBc`
- If `(bassEnhType == uhAbassEnh_PhantomBass)`
  - `abe.minEffectStrengthPb <= effectStrength <= abe.maxEffectStrengthPb`
- If `(bassEnhType == uhAbassEnh_DynamicBassEnhancement)`
  - `abe.minEffectStrengthDbe <= effectStrength <= abe.maxEffectStrengthDbe`

Action
- `Modify abe.currentBassEnhTypes`
- If `(bassEnhType == uhAbassEnh_BassCompensation)`
  - Modify `abe.effectStrengthBc`
- If `(bassEnhType == uhAbassEnh_PhantomBass)`
  - Modify `abe.effectStrengthPb`
- If `(bassEnhType == uhAbassEnh_DynamicBassEnhancement)`
  - Modify `abe.effectStrengthDbe`

Postcondition
- If `(enable == True)`
  - `abe.currentBassEnhTypes & bassEnhType != 0`

  else
  - `abe.currentBassEnhTypes & bassEnhType == 0`
- If `(bassEnhType == uhAbassEnh_BassCompensation)`
  - `abe.effectStrengthBc == effectStrength`
- If `(bassEnhType == uhAbassEnh_PhantomBass)`
  - `abe.effectStrengthPb == effectStrength`
- If `(bassEnhType == uhAbassEnh_DynamicBassEnhancement)`
  - `abe.effectStrengthDbe == effectStrength`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.2.5.1.4 GetBassEnh

Signature

```
uhErrorCode_t GetBassEnh (
    [in] uhAbassEnh_Type_t bassEnhType,
    [out] pInt32 pEffectStrength,
    [out] pBool pEnabled);
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns the “effect-strength” and current activation status for a given bass enhancement type.

Parameters

Table 352

Name	Description
`bassEnhType`	The bass enhancement type for which the “effect-strength” and the activation state are queried.
`pEffectStrength`	Output parameter returning the “effect-strength” for the bass enhancement type `bassEnhType`. It is specified in integral steps within the range corresponding to `bassEnhType`.
`pEnabled`	Output parameter returning the current activation state for the bass enhancement type `bassEnhType`. It is `True` when the specified bass enhancement type is active. It is `False` when the specified bass enhancement type is not active.

Return Values

Standard.

Precondition

```
bassEnhType & abe.suppBassEnhTypes != 0
```

Action

None.

Postcondition

— If `(bassEnhType & abe.currentBassEnhTypes == 0)`
  — `*pEnabled == False`
  `else`
  — `*pEnabled == True`
— If `(bassEnhType == uhAbassEnh_BassCompensation)`
  — `*pEffectStrength == abe.effectStrengthBc`
— If `(bassEnhType == uhAbassEnh_PhantomBass)`
  — `*pEffectStrength == abe.effectStrengthPb`
— If `(bassEnhType == uhAbassEnh_DynamicBassEnhancement)`
  — `*pEffectStrength == abe.effectStrengthDbe`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.3 UhlAdrc

7.2.3.1 Concepts

7.2.3.1.1 Introduction

Audio DRC (ADRC) reduces the dynamic range of the audio signal to get low-level and high-level signals closer together in level. The signal can thus be adapted to the reproduction apparatus and/or the listening conditions. DRC can e.g. be used to attenuate (or 'limit') high-level signals while passing low-level signals unchanged, or to boost low-level signals while passing high-level signals unchanged, or to attenuate high-level signals and boost low-level signals. An example is the so-called "midnight mode" of operation for TVs that allows reducing the overall volume of the program without significantly affecting the intelligibility of the dialogues.

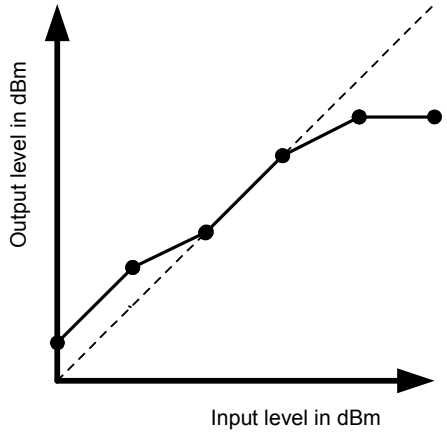


Figure 39 — A typical curve for dynamic range control. The dots on the curve denote the "elbow points"

DRC operation can be precisely specified through a curve as illustrated in Figure 39. For compressed audio, the requirement can also be met through a gain parameter associated with each frame of audio data. Different gain parameters may be included in a single stream corresponding to different DRC settings.

The gain parameter is frequently associated with the audio stream as meta-data. For example, MPEG-1 [1] and MPEG-2 [2] used for DVB, DVD-Video etc. optionally carries the meta-data inside the 'ancillary data' field of the coded audio bit-stream. AC-3 audio, used for the ATSC TV system [13] and DVD-Video, carries the meta-data in fields 'dynrng' and 'compr' in the coded audio bitstream. As indicated p. 4-20 of the AC-3 Licensee Information Manual [22] Dynamic Range Compression is required in all Dolby Digital products, it is also available for DTS streams.

7.2.3.1.2 Operational Mode for ADRC

Control is provided to the user to query what DRC modes (light/medium/heavy etc.) are supported, and to set a preference for a DRC mode from among this set.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Depending on the signal properties, it may not be advisable to apply DRC or it might be necessary to adapt the selected DRC mode accordingly.

Wherever such possibilities of conflict arise, the DRC functionality may be autonomously be shut off or the DRC mode applied may be autonomously enforced to an other mode.

In case a DRC mode is not available for a particular stream, the DRC mode enforced by the LC will be different than the mode selected by the user, based on availability of DRC options. The closest supported mode will be autonomously substitued to it. See Table 353 for details of the mode mapping that the user can expect from the LC, in case substitution is applied.

Table 353 — DRC mode mapping

Supported and Available Mode(s) for the current audio input stream	(User) Selected Mode	Actual Mode
Invalid input signal	No/Light/Medium/Heavy Compression	Invalid
Unknown input signal	No/Light/Medium/Heavy Compression	Unknown
No Compression	No/Light/Medium/Heavy Compression	No Compression
No Compression,	No Compression	No Compression
Light Compression	Light/Medium/Heavy Compression	Light Compression
No Compression,	No Compression	No Compression
Medium Compression	Light/Medium/Heavy Compression	Medium Compression
No Compression,	No Compression	No Compression
Heavy Compression	Light/Medium/Heavy Compression	Heavy Compression
No Compression,	No Compression	No Compression
Light Compression,	Light/Medium Compression	Light Compression
Heavy Compression	Heavy Compression	Heavy Compression
No Compression,	No Compression	No Compression
Light Compression,	Light Compression	Light Compression
Medium Compression	Medium/Heavy Compression	Medium Compression
No Compression,	No Compression	No Compression
Medium Compression,	Light/Medium Compression	Medium Compression
Heavy Compression	Heavy Compression	Heavy Compression
No Compression,	No Compression	No Compression
Light Compression,	Light Compression	Light Compression
Medium Compression,	Medium Compression	Medium Compression
Heavy Compression	Heavy Compression	Heavy Compression

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For practical purposes, the "light", "medium" and "heavy" modes constitute a set of monotonically increasing DRC effect qualifiers.

7.2.3.2 Types & Constants

7.2.3.2.1 Public Types & Constants

7.2.3.2.1.1 Error Codes

None.

7.2.3.2.1.2 uhAdrc_Mode_t

Signature

```
typedef enum _uhAdrc_Mode_t {
    uhAdrc_ModeInvalid     = 0x80000000,
    uhAdrc_ModeUnknown     = 0x40000000,
    uhAdrc_ModeNoDrc       = 0x00000001,
    uhAdrc_ModeLightCompr  = 0x00000002,
    uhAdrc_ModeMediumCompr = 0x00000004,
    uhAdrc_ModeHeavyCompr  = 0x00000008
} uhAdrc_Mode_t, *puhAdrc_Mode_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the DRC modes settable by the user (see subclause 7.2.3.1.2).

Values

Table 354

Name	Description
uhAdrc_ModeInvalid	The information over the DRC data is temporarily not available.
uhAdrc_ModeUnknown	The information over the DRC data is available, yet the mode could not be determined.
uhAdrc_ModeNoDrc	DRC is not applied on the input signal. The output stream is identical to the input stream.
uhAdrc_ModeLightCompr	Generic light dynamic range compression mode, where low-level signals are enhanced, and very loud sounds reduced in volume.
uhAdrc_ModeMediumCompr	Generic medium dynamic range compression mode, somewhere between "light" and "heavy".
uhAdrc_ModeHeavyCompr	Generic heavy dynamic range compression mode, suitable for use as the "night mode". The difference between low and high output audio levels reduces further from the "light" and "medium" modes.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.3.2.1.3 uhAdrc_ModeSet_t

Signature
```
typedef Int32
uhAdrc_ModeSet_t, *p
uhAdrc_ModeSet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of ADRC modes.

Constraints
None.

7.2.3.2.1.4 uhAdrc_Ntf_t

Signature
```
typedef enum _uhAdrc_Ntf_t {
    uhAdrc_SubscriptionChanged    = 0x00000001,
    uhAdrc_ActualModeChanged      = 0x00000002
} uhAdrc_Ntf_t, *puhAdrc_Ntf_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAdrcNtf`.

Values

Table 355

Name	Description
`uhAdrc_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhAdrc_ActualModeChanged`	Represents `OnActEnabledChanged`.

7.2.3.2.1.5 uhAdrc_NtfSet_t

Signature
```
typedef Int32 uhAdrc_NtfSet_t, *puhAdrc_NtfSet_t;
```

Qualifiers
— enum-set

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhAdrc_Ntf_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

Constraints
None.

7.2.3.2.1.6 uhAdrc_AllNtfs

Signature
```
const uhAdrc_NtfSet_t uhAdrc_AllNtfs = 0x00000003;
```

Qualifiers
None.

Description
Defines the set of all values of type uhAdrc_Ntf_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

7.2.3.2.2 Model Types & Constants

None.

7.2.3.3 Logical Component

7.2.3.3.1 Interface-Role Model

Figure 40 depicts the interface-role model of the ADRC logical component. It shows the interfaces and the roles involved with this component.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

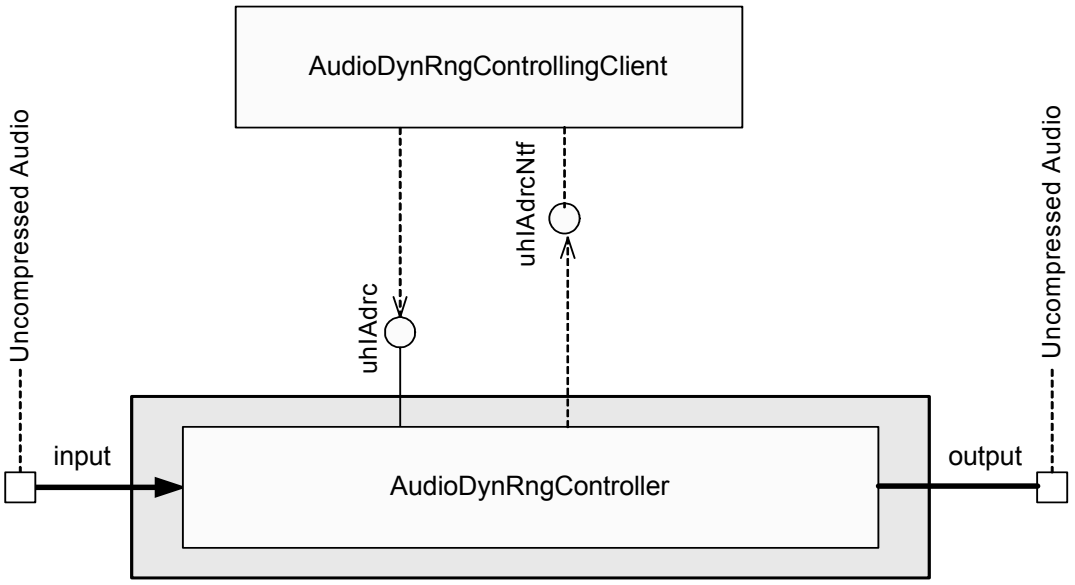


Figure 40 — Interface Role Model

The grey box in Figure 40 denotes an Audio Dynamic Range Control logical component containing a single `AudioDynRngController` role.

This role provides one mandatory (solid line) control interface called `UhIAdrc` to its client, the `AudioDynRngControllingClient`.

This client can optionally provide a call-back interface (`uhIAdrcNtf`) allowing it to receive notifications of changes in the actual DRC mode applied to the audio input stream.

7.2.3.3.2 Diversity

7.2.3.3.2.1 Provided Interfaces

Table 356

Role	Interface	Presence
`AudioDynRngController`	`UhIAdrc`	mandatory
`AudioDynRngControllingClient`	`uhIAdrcNtf`	optional

7.2.3.3.2.2 Configurable Items

Table 357

Role	Attribute
`AudioDynRngController`	`suppModes`

7.2.3.3.2.3 Constraints

— `!( suppModes & uhAdrc_ModeInvalid)`
— `!( suppModes & uhAdrc_ModeUnknown)`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— (suppModes & uhAdrc_ModeNoDrc) == uhAdrc_ModeNoDrc
— suppModes != uhAdrc_ModeNoDrc

7.2.3.3.3 Instantiation

7.2.3.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 358

Type	Object	Multiplicity
AudioDynRngController	drc	1

7.2.3.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
drc.prefMode == uhAdrc_ModeNoDrc
```

```
drc.actualMode == uhAdrc_ModeInvalid
```

7.2.3.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

7.2.3.4 Roles

7.2.3.4.1 AudioDynRngController

Signature

```
role AudioDynRngController {
    const
uhAdrc_ModeSet_t  suppModes;
    uhAdrc_Mode_t           prefMode;
    uhAdrc_Mode_t           actualMode;
}
```

Qualifiers

— root

Description

The AudioDynRngController role performs the function of accepting DRC settings from the user, and executing DRC operation on the input stream in a way as to fit user preferences.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 359

Name	Description
suppModes	The set of supported DRC modes.
prefMode	The user preference for the DRC mode that should be applied to the audio input stream.
actualMode	The actual DRC mode applied to the audio input stream.

Dependent Attributes
None.

Invariants

```
— (suppModes & prefMode) == prefMode
— if ( (actualMode != uhAdrc_ModeInvalid) || (actualMode != uhAdrc_ModeUnknown) )
  — (suppModes & actualMode) == actualMode
```

Instantiation
An AudioDynRngController is always instantiated as part of an Audio Dynamic Range Control LC; see subclause 7.2.3.3.3.

Streaming Behavior
The AudioDynRngController has an uncompressed audio stream as input. It processes the audio signal according to the DRC mode in actualMode. This processed audio signal is passed to the output.

When the actual DRC mode is uhAdrc_NoDrcMode, the stream at the input is passed to the output without any processing.

Active Behavior
The main events associated with the AudioDynRngController role are as follows:

Table 360

Event	Action
A modification in the DRC mode that is applied to the audio input stream occurs related to: — Either a modification in terms of availability of the DRC modes due to a change in the stream format or in the stream's metadata; — Either a modification of the user preference for the DRC mode that should preferably be applied.	OnActualModeChanged

Execution Constraints
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.3.4.2 AudioDynRngControllingClient

Signature
```
role AudioDynRngControllingClient {}
```

Qualifiers
— actor

Description
An `AudioDynRngControllingClient` represents a user of the `UhIAdrc` interface.

Execution Constraints
None.

7.2.3.5 Interfaces

7.2.3.5.1 uhIAdrc

Qualifiers
None.

Description
This interface of an ADRC provides functions to set and retrieve DRC preferences.

Interface ID
```
uuid( C4E4ABE5-7281-4D18-A82E-DD6CCE1FEA7B )
```

7.2.3.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAdrcNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAdrc_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.2.3.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAdrcNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAdrc_NtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.3.5.1.3 GetSuppModes

Signature
```
uhErrorCode_t GetSuppModes ( [out] p
uhAdrc_ModeSet_t pSuppModes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the set of DRC modes supported by the ADRC.

Parameters

Table 361

Name	Description
`pSuppModes`	Output parameter returning the set of DRC modes that the ADRC supports.

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
*pSuppModes == drc.suppModes
```

7.2.3.5.1.4 SetMode

Signature
```
uhErrorCode_t SetMode ( [in] uhAdrc_Mode_t mode );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the DRC mode that should preferably be applied to the audio input stream.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 362

Name	Description
`mode`	The user preference for the DRC mode that should be applied to the audio input stream.

Return Values
Standard.

Precondition
`(mode & drc.suppModes) != 0`

Action
— Modify `drc.prefMode`

Postcondition
`drc.prefMode == mode`

7.2.3.5.1.5 GetMode

Signature
`uhErrorCode_t GetMode ( [out] puhAdrc_Mode_t pMode );`

Qualifiers
— synchronous

— thread-safe

Description
Returns the DRC mode that should preferably be applied to the audio input stream.

Parameters

Table 363

Name	Description
`pMode`	Output parameter returning the user preference for the DRC mode that should be applied to the audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pMode == drc.prefMode`

7.2.3.5.1.6 GetActualMode

Signature
```
uhErrorCode_t GetActualMode ( [out] puhAdrc_Mode_t pActualMode );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the actual DRC mode applied to the audio input stream.

Parameters

Table 364

Name	Description
`pActualMode`	Output parameter returning the actual DRC mode applied to the audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pActualMode == drc.actualMode`

7.2.3.5.2 uhIAdrcNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of changes in the actual DRC mode applied to the input audio stream. The `Subscribe` function of the `UhIAdrc` interface can be used to subscribe to these events.

Interface ID
```
uuid( 0BAC03CE-BFB7-412b-946B-E332BED95F3B )
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.3.5.2.1 OnSubscriptionChanged

Signature

```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAdrc_NtfSet_t notifs );
```

Qualifiers

— onsubscriptionchanged-function

7.2.3.5.2.2 OnActualModeChanged

Signature

```
[async] Void OnActualModeChanged (
    [in] UInt32 cookie,
    [in] uhAdrc_Mode_t actualMode );
```

Qualifiers

None.

Description

This function is called by the `AudioDynRngController` whenever the actual DRC mode applied to the audio input stream has changed.

Parameters

Table 365

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`actualMode`	The actual DRC mode that is applied to the audio input stream.

Return Values

None.

Precondition

`True`

Action

— Any callback-compliant action

Postcondition

`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.4 UhlAmix

7.2.4.1 Concepts

7.2.4.1.1 Introduction

An Audio Mixer is able to sum uncompressed baseband audio input streams into a single baseband audio output stream.

Figure 41 — General structure of an Audio Mixer shows the general structure of an Audio Mixer.

An Audio Mixer has two or more uncompressed baseband audio input signals (three in the case of Figure 41 — General structure of an Audio Mixer). Each input signal has its own Gain and Mute Control. The adder sums the inputs together and the result of the mixing is a single baseband audio signal.

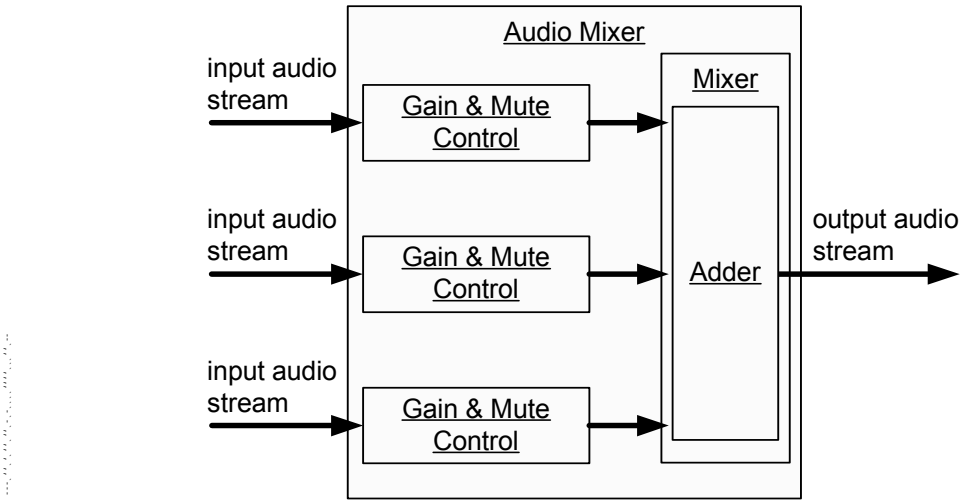


Figure 41 — General structure of an Audio Mixer

If the same input signals must be mixed to different output signals, e.g. a complete karaoke mix with vocals and a karaoke mix without vocals, then two different Audio Mixers must be used.

The Audio Mixer should not be confused with a downmixer. The Audio Mixer mixes together two or more audio signals (each containing one or more audio channels), while a downmixer has as input a single audio signal with multiple audio channels, which it downmixes to an audio signal with less audio channels than its input.

7.2.4.1.2 Gain & Mute Control

A Gain & Mute Control provides to its client(s) functionality to explicitly mute or un-mute its input signal.

Mute control is effective at the audio signal level, i.e. all the audio channels of the input signal are either muted, either un-muted.

Muting and gain operate independently; setting a gain level doesn't unmute the audio (if muted). During muting the gain setting maintains its value. When the audio is unmuted this value is then applied.

Gain control is also effective at the audio signal level, i.e. the gain value is applied to all the audio channels of the input signal.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.4.1.3 Mixing of the input streams

The Mixers function is to add the audio input signals together.

The Mixer adds matching audio channels together, i.e. left channels are added to left channels, right channels are added to right channels, etc. The output signal will have the superset of the audio channels of the inputs; see Figure 42 — Mixing input signals with different channel configurations.

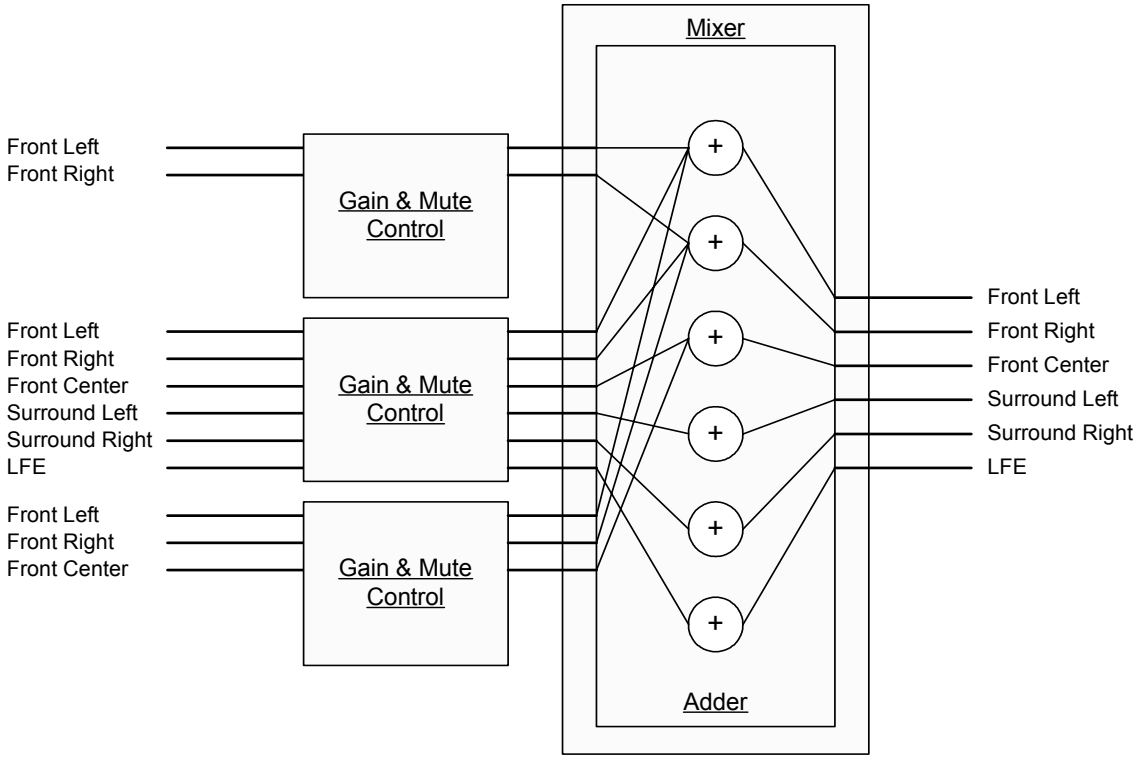


Figure 42 — Mixing input signals with different channel configurations

When one of the audio input streams of the Audio Mixer contains a mono signal (i.e. only a single front channel) then three cases can be identified:

— At least one of the other audio input streams of the Audio Mixer contains a front center channel, then the mono signal will be added to the front center channel of the other (relevant) audio input stream(s); see Figure 43;

— None of the audio input streams of the Audio Mixer contains a front center channel, but at least one stream contains a front right and a front left channel, then the mono signal will be added after proper scaling (for total energy preservation) to the front left and to the front right channels of the other (relevant) audio input stream(s); see Figure 44;

— All the audio input streams of the Audio Mixer only contain a single front channel then all the (mono) signals are added together; see Figure 45.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

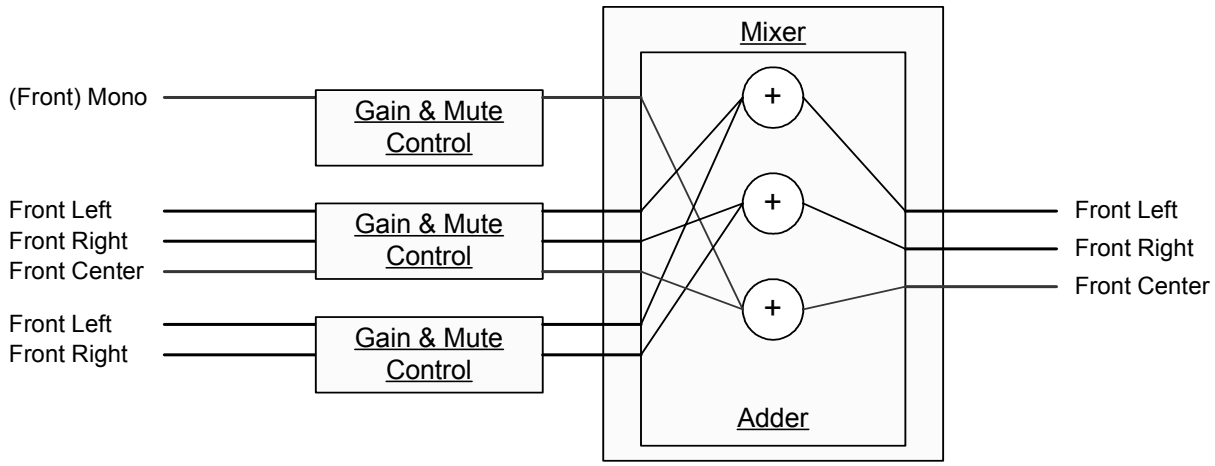


Figure 43 — Mixing input signals when (at least) one of the input streams contains a (front) mono signal and (at least) one of the input streams contains a front center signal.

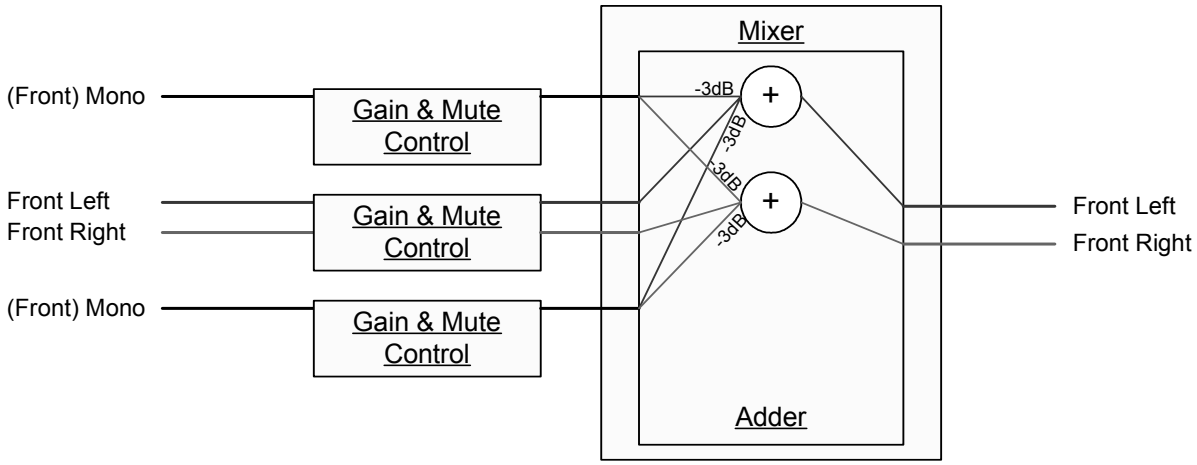


Figure 44 — Mixing input signals when (at least) one of the input streams contains a (front) mono signal, (at least) one of the input streams contains stereo front signals but none of the input streams contains a front center signal.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

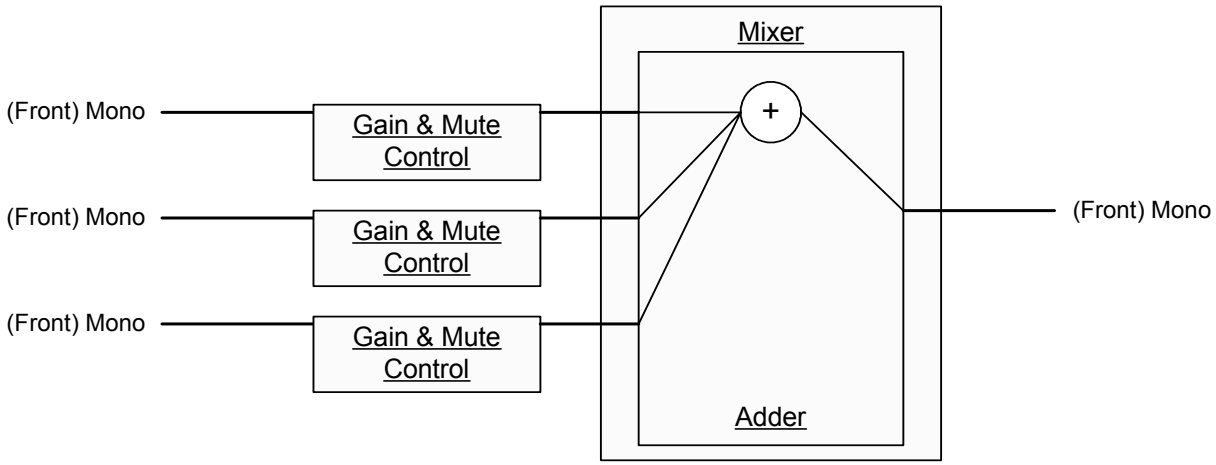


Figure 45 — Mixing input signals when all the input streams contain a (front) mono signal.

In the same way, when one of the audio input stream of the Audio Mixer contains a single surround (i.e. rear) channel then three cases can be identified:

— At least one of the other audio input streams of the Audio Mixer contains a surround center channel, then the surround mono signal will be added to the surround center channel of the other (relevant) audio input stream(s); see Figure 46;

— None of the audio input streams of the Audio Mixer contains a surround center channel, but at least one stream contains a surround right and a surround left channel, then the surround mono signal will be added after proper scaling (for total energy preservation) to the surround left and to the surround right channels of the other (relevant) audio input stream(s); see Figure 47;

— None of the audio input streams of the Audio Mixer contains either a surround center channel nor a surround right and a surround left channel, then all the (mono) surround signals are added together; see Figure 48.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

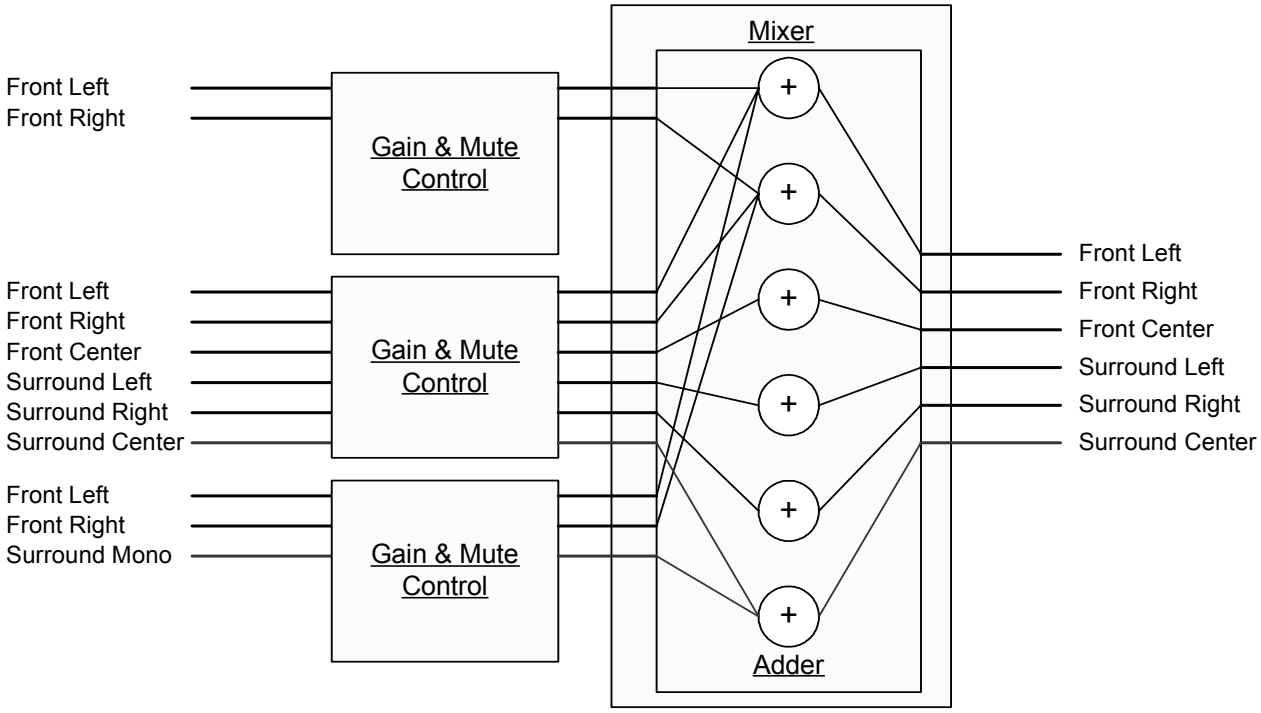


Figure 46 — Mixing input signals when (at least) one of the input streams contains a surround mono signal and (at least) one of the input streams contains a surround center signal.

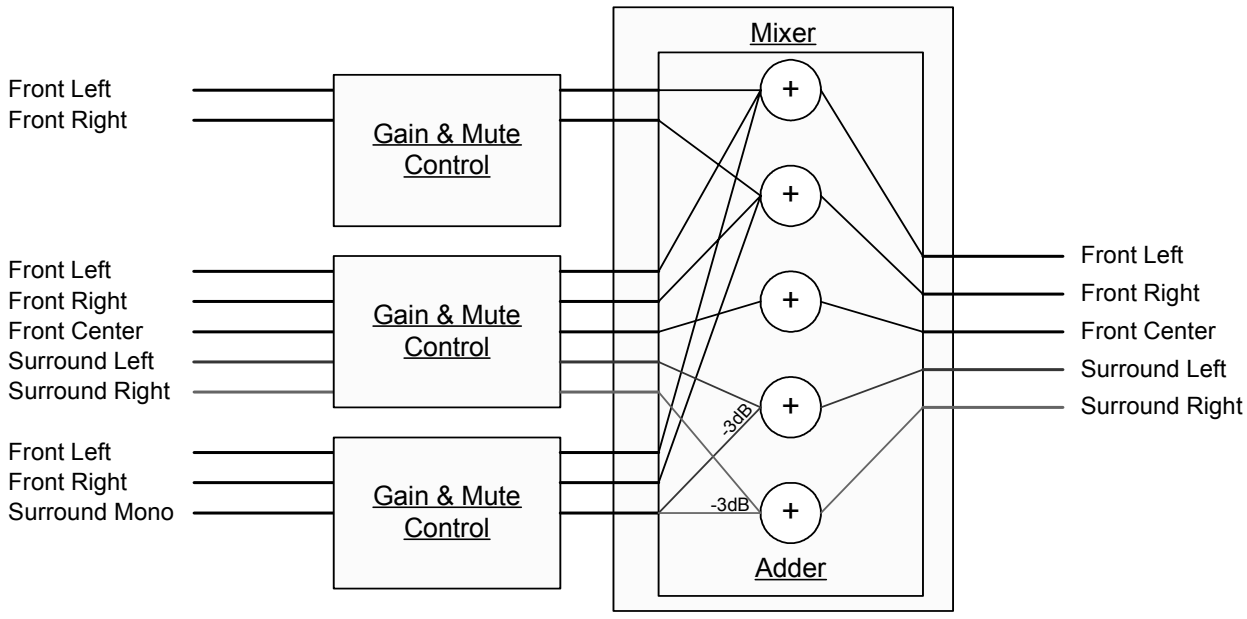


Figure 47 — Mixing input signals when (at least) one of the input streams contains a surround mono signal, (at least) one of the input streams contains stereo surround signals but none of the input streams contains a surround center signal.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

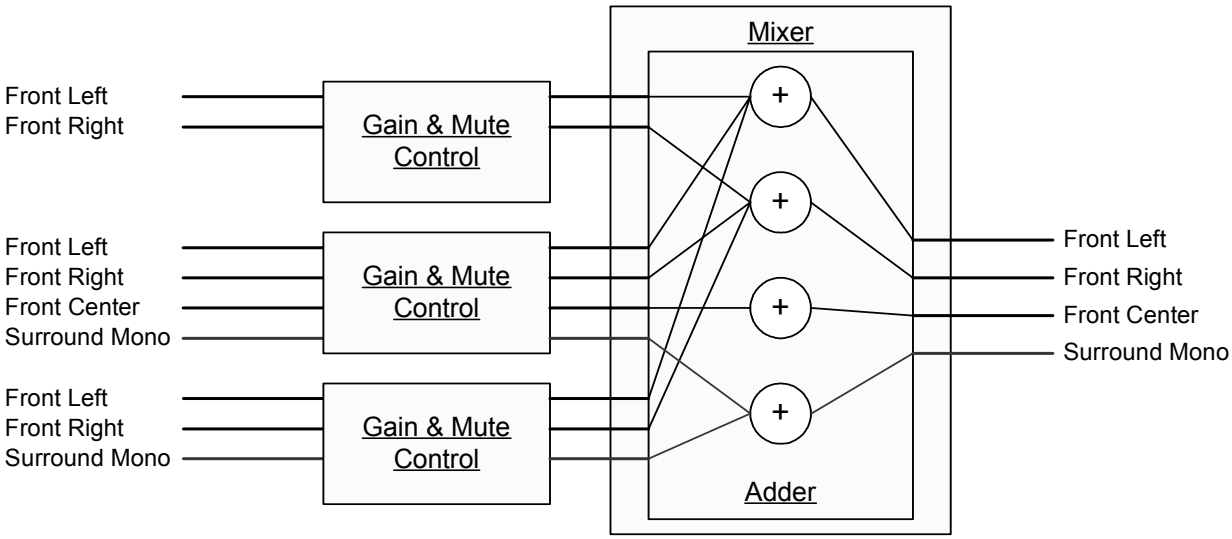


Figure 48 — Mixing input signals when the input streams contain at most a surround mono signal.

7.2.4.2 Types & Constants

7.2.4.2.1 Public Types & Constants

7.2.4.2.1.1 Error Codes

None.

7.2.4.2.1.2 uhAmix_InBase

Signature

```
const UInt32 uhAmix_InBase = 0;
```

Qualifiers
None.

Description
Defines the constant that identifies the first input pin of an Audio Mixer.

7.2.4.2.2 Model Types & Constants

None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.4.3 Logical Component

7.2.4.3.1 Interface-Role Model

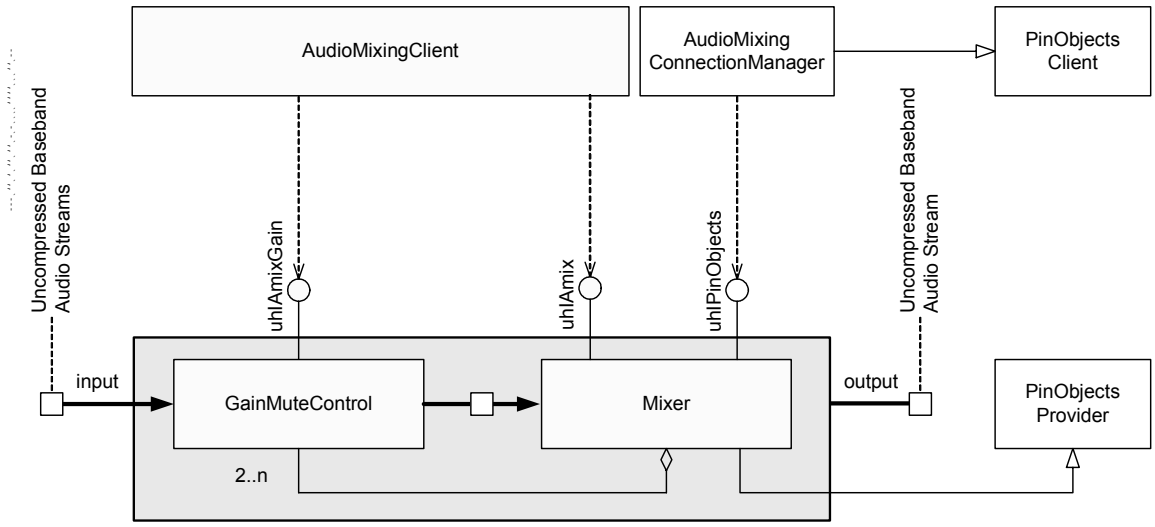


Figure 49 — Interface Role Model

An Audio Mixing Logical Component (the grey box in the Interface Role diagram) contains a Mixer (represented by the `Mixer` role) and two or more GainMuteControls (represented by the `GainMuteControl` role).

The GainMuteControls are part of the Mixer.

Each GainMuteControl has its own control interface (`uhIAmix`). This control interface is meant for use by the regular client (represented by the `AudioMixingClient` role).

Each Mixer has an interface (`uhIPinObjects`) that maps the pin ID of an audio input stream to the GainMuteControl processing this input stream. This interface is meant for exclusive use by the connection manager (represented by the `AudioMixingConnectionManager` role). The `uhIAmixGain` interface of a GainMuteControl can be obtained via the `GetPinRelatedObject` function of the `uhIPinObjects` interface; see 6.1.5.

The "base roles" `PinObjectsClient` and `PinObjectsProvider` are defined in 6.1.5.

7.2.4.3.2 Diversity

7.2.4.3.2.1 Provided Interfaces

Table 366

Role	Interface	Presence
`Mixer`	`uhIAmix`	mandatory
	`uhIPinObjects`	mandatory
`GainMuteControl`	`uhIAmixGain`	mandatory

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.4.3.2.2 Configurable Items

Table 367

Role	Attribute
Mixer	nrGmControls
GainMuteControl	minGain
	maxGain
	unityGain
	muteSupported

7.2.4.3.2.3 Constraints

— nrGmControls ≥ 2
— For each i in {0 .. mix.nrGmControls-1}
  — mix.gmc[i].minGain ≤ mix.gmc[i].unityGain ≤ mix.gmc[i].maxGain
— For i & j in {0 .. mix.nrGmControls-1}
  — mix.gmc[i].minGain == mix.gmc[j].minGain
  — mix.gmc[i].maxGain == mix.gmc[j].maxGain
  — mix.gmc[i].unityGain == mix.gmc[j].unityGain

7.2.4.3.3 Instantiation

7.2.4.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 368

Type	Object	Multiplicity
Mixer	mix	1
GainMuteControl	gmc[]	mix.nrGmControls

7.2.4.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— mix.pinIds == mix.inputPinIds
— For each i in {0 .. mix.nrGmControls-1}
  — mix.pinObject[uhAmix_InBase+i] == gmc[i]
  — mix.gmControls == {gmc[0] .. gmc[mix.nrGmControls-1]}
— For each i in {0 .. mix.nrGmControls-1}
  — gmc[i].mixer == mix
  — gmc[i].gain == gmc[i].unityGain
— If (gmc[i].muteSupported)
  — gmc[i].mute == True
  else
  — gmc[i].mute == False

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.4.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

7.2.4.4 Roles

7.2.4.4.1 Mixer

Signature

```
role Mixer specializes PinObjectsProvider {
    const UInt32                    nrGmControls;
    const setof{GainMuteControl}    gmControls;
    const setof{PinId}              inputPinIds;
}
```

Qualifiers

— root

Description

A Mixer role adds its multiple audio input signals into a single audio signal, matching the audio channels together.

For connection management purposes, the Mixer maintains the relation between the input ids of the Audio Mixer and the GainMuteControls associated with them.

Independent Attributes

Table 369

Name	Description
`nrGmControls`	The number of GainMuteControls associated with the Mixer.
`gmControls`	The set of GainMuteControls associated with the Mixer.

Dependent Attributes

Table 370

Name	Description
`inputPinIds`	The set of pin ids of the GainMuteControls associated with the Mixer, which is equal to { `uhAmix_InBase` + `i` for which `0` ≤ `i` < `nrGmControls` }.

Invariants

None.

Instantiation

A Mixer is always instantiated as part of an Audio Mixer; see subclause 7.2.4.3.3.

Streaming Behavior

The Mixer has `nrGmControls` audio input streams. It adds these multiple audio signals into a single audio signal, matching the channels together and outputs a single audio stream. See subclause 7.2.4.1.3 for details about the specifics of mixing when at least one input stream contains a single front or a single surround (i.e. rear) audio channel.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Active Behavior
None.

Execution Constraints
None.

Remarks

— The constant attributes `pinIds` and `pinObject` are inherited from the `PinObjectProvider` role defined in 6.1.5 which specifies no values for these attributes; see subclause 7.2.4.3.3.2 for the initial value of `pinIds` and `pinObject`.

7.2.4.4.2 GainMuteControl

Signature

```
role GainMuteControl {
    const Mixer                 mixer;
    const Int32                 minGain;
    const Int32                 maxGain;
    const Int32                 unityGain;
    Int32                       gain;
    const Bool                  muteSupported;
    Bool                        mute;
}
```

Qualifiers
None.

Description
A GainMuteControl provides the following functionality:

— Muting and unmuting its audio input stream, if supported;

— Applying a gain to its audio input stream.

Independent Attributes

Table 371

Name	Description
`mixer`	The Mixer with which this GainMuteControl is associated.
`minGain`	The minimum gain that can be applied to the audio input stream of the GainMuteControl. Minimum gain is not necessarily equivalent to mute activated, at minimum gain an audio signal can still be audible.
`maxGain`	The maximum gain that can be applied to the audio input stream of the GainMuteControl.
`unityGain`	The unity gain of the GainMuteControl, for which the audio input stream is passed on unchanged, i.e. neither amplified, nor attenuated.
`gain`	The gain applied to the audio input stream of the GainMuteControl, when the audio input stream of the GainMuteControl is not muted.
`muteSupported`	Describes whether mute control is supported by the GainMuteControl.
`mute`	Describes whether the input stream of the GainMuteControl is muted or not.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes
None.

Invariants
— `minGain ≤ gain ≤ maxGain`
— If `(!muteSupported)`
  — `mute == False`

Instantiation
A GainMuteControl is always instantiated as part of an Audio Mixer; see subclause 7.2.4.3.3.

Streaming Behavior
A GainMuteControl has an audio signal as input stream. When its gain (`gain`) is different from the unity gain (`unityGain`), the GainMuteControl attenuates the audio signal if `gain < unityGain` or amplifies the audio signal if `gain > unityGain`.

No output data is produced when the GainMuteControl is muted.

The GainMuteControl takes care of a "smooth" transition while muting or un-muting, meaning that audible artefacts like "clicks" do not occur.

Active Behavior
None.

Execution Constraints
None.

7.2.4.4.3 AudioMixingClient

Signature
`role` `AudioMixingClient {}`

Qualifiers
— actor

Description
An Audio Mixing Client represents a user of the `uhIAmix` and `uhIAmixGain` interfaces.

Execution Constraints
None.

7.2.4.4.4 AudioMixingConnectionManager

Signature
`role` `AudioMixingConnectionManager` **`specializes`** `PinObjectsClient {}`

Qualifiers
— actor

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
An Audio Mixing Connection Manager represents an user of the `uhIPinObjects` interface, which will normally be the connection manager responsible for establishing signal paths between logical components.

Execution Constraints
None.

7.2.4.5 Interfaces

7.2.4.5.1 Mixer::uhIPinObjects

Qualifiers
None.

Description
This interface of a Mixer is a specialization of the `uhIPinObjects` interface as defined in 6.1.5. The specialization is defined by the specific values of the `pinIds` and `pinObject` attributes of the Mixer. These attributes are inherited from `PinObjectsProvider` and their values are set at instantiation time of a Mixer; see subclause 7.2.4.3.3.2.

Interface ID
See 6.1.5.

7.2.4.5.2 uhIAmix

Qualifiers
None.

Description
This interface of a Mixer provides functionality to query the number of GainMuteControls it supports.

Interface ID
`uuid( 454B03DF-A493-4573-86C2-8B21D1070CDB )`

7.2.4.5.2.1 GetSuppNrControls

Signature
```
uhErrorCode_t GetSuppNrControls ( [out] pUInt32 pNrControls );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of GainMuteControls supported by the Audio Mixer.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 372

Name	Description
pNrControls	Output parameter returning the number of GainMuteControls supported by the Audio Mixer.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pNrControls == mix.nrGmControls

Remarks

— The number of GainMuteControls will be at least 2.

7.2.4.5.3 uhIAmixGain

Qualifiers
None.

Description
This interface of a GainMuteControl provides functionality to control the gain applied to the audio input stream. It also provides functionality to mute the audio input stream.

Interface ID
uuid(D209FD3D-8D0C-481D-AA8A-EA401E8259F6)

7.2.4.5.3.1 GetGainRange

Signature
```
uhErrorCode_t GetGainRange (
    [out] pInt32 pMinGain,
    [out] pInt32 pMaxGain );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the minimum and the maximum gain that can be applied to the audio input stream of the GainMuteControl.

Parameters

Table 373

Name	Description
`pMinGain`	Output parameter returning the minimum gain that can be applied to the audio input stream of the GainMuteControl.
`pMaxGain`	Output parameter returning the maximum gain that can be applied to the audio input stream of the GainMuteControl.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `gmc` be the `GainMuteControl` object on which this interface operates
— `*pMinGain == gmc.minGain`
— `*pMaxGain == gmc.maxGain`

7.2.4.5.3.2 GetUnityGain

Signature
```
uhErrorCode_t GetUnityGain ( [out] pInt32 pUnityGain );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the unity gain.

Parameters

Table 374

Name	Description
`pUnityGain`	Output parameter returning the unity gain of the GainMuteControl.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
— Let `gmc` be the `GainMuteControl` object on which this interface operates
— `*pUnityGain == gmc.unityGain`

Remarks
— At unity gain the audio signal is passed on unchanged: the audio signal is neither amplified, nor attenuated.

7.2.4.5.3.3 SetGain

Signature
```
uhErrorCode_t SetGain ( [in] Int32 gain );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the gain that should be applied to the audio input stream of the GainMuteControl when the audio input stream of the GainMuteControl is not muted.

Parameters

Table 375

Name	Description
`gain`	Indicates the gain that should be applied to the audio input stream of the GainMuteControl when the audio input stream of the GainMuteControl is not muted.

Return Values
Standard.

Precondition
— Let `gmc` be the `GainMuteControl object` on which this interface operates
— `gmc.minGain ≤ gain ≤ gmc.maxGain`

Action
— Modify `gmc.gain`

Postcondition
`gmc.gain == gain`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.4.5.3.4 GetGain

Signature
`uhErrorCode_t GetGain ( [out] pInt32 pGain );`

Qualifiers
— synchronous

— thread-safe

Description
Returns the gain that should be applied to the audio input stream of the GainMuteControl when the audio input stream of the GainMuteControl is not muted.

Parameters

Table 376

Name	Description
`pGain`	Output parameter returning the gain that should be applied to the audio input stream of the GainMuteControl when the audio input stream of the GainMuteControl is not muted.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `gmc` be the `GainMuteControl` object on which this interface operates
— `*pGain == gmc.gain`

7.2.4.5.3.5 GetMuteSupported

Signature
`uhErrorCode_t GetMuteSupported ( [out] pBool pMuteSupported );`

Qualifiers
— synchronous

— thread-safe

Description
Returns whether mute control is supported by the GainMuteControl.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 377

Name	Description
pMuteSupported	Output parameter returning whether mute control is supported by the GainMuteControl.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— Let gmc be the GainMuteControl object on which this interface operates
— *pMuteSupported == gmc.muteSupported

7.2.4.5.3.6 SetMute

Signature
```
uhErrorCode_t SetMute ( [in] Bool mute );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets whether the audio input stream of the GainMuteControl should be muted or not.

If the GainMuteControl is muted, no output data is produced.

The GainMuteControl takes care of a "smooth" transition while muting or un-muting, meaning that audible artefacts like "clicks" do not occur.

Parameters

Table 378

Name	Description
mute	Defines whether the audio input stream of the GainMuteControl should be muted or not. It is True when the audio input stream should be muted. It is False when the audio input stream should not be muted.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
— Let `gmc` be the `GainMuteControl object` on which this interface operates
— `gmc.supportedMute == True`

Action
— Modify `gmc.mute`

Postcondition
`gmc.mute == mute`

7.2.4.5.3.7 GetMuted

Signature
```
uhErrorCode_t GetMuted ( [out] pBool pMuted );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the audio input stream of the GainMuteControl is currently muted or not.

Parameters

Table 379

Name	Description
`pMuted`	Output parameter that indicates whether the audio input stream of the GainMuteControl is currently muted or not. It is `True` when the audio input stream is muted. It is `False` when the audio input stream is not muted.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `gmc` be the `GainMuteControl` object on which this interface operates
— `*pMuted == gmc.mute`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.5 UhlAnoiseGen

7.2.5.1 Concepts

7.2.5.1.1 Introduction

Noise generation (NG) is used for setting speaker trim levels at the time of installation[9] to position the speaker in the audio sweet spot (region of best surround effects). The logical model for an ANG comprises a filter, which switches between the input audio data and a noise source depending on whether the generation functionality is activated. In the latter case, the noise signal is generated on one output channel at a time, while the others are muted. When the noise generator is switched off, the LC functions as a "through" component, meaning that the audio stream at the input is replicated at the output. The behaviour is illustrated in the following figure.

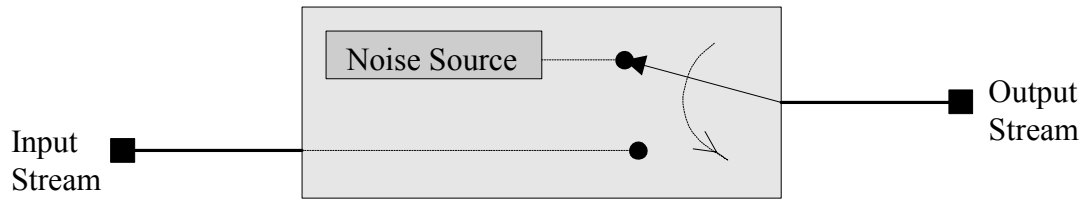


Figure 50 — Conceptual block diagram for the noise generator.

7.2.5.1.2 Operation Mode for ANG

The ANG can function in the "manual" mode only, meaning that the user has to specify the channel on which noise is generated on being switched on. Timing management for the duration of noise generation has to be performed by the user. The channel for noise generation can be selected by the user before activating noise generation, or during operation in the active mode.

7.2.5.1.3 Noise Characteristics

The noise generated by the ANG usually is centered around a given frequency, which is selectable by the user. The slope and bandwidth of the spectral wave-shape of the "noise" signal is usually platform dependent, and the specifics will not be programmable by the user.

9 More details about the use of noise generators can be found in reference [6].

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.5.2 Types & Constants

7.2.5.2.1 Public Types & Constants

7.2.5.2.1.1 uhAnoiseGen_NoiseType_t

Signature

```
typedef enum _uhAnoiseGen_NoiseType_t {
    uhAnoiseGen_Woofer  = 0x00000001,
    uhAnoiseGen_Normal  = 0x00000002,
    uhAnoiseGen_Tweeter = 0x00000004
} uhAnoiseGen_NoiseType_t, *puhAnoiseGen_NoiseType_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the different kinds of noise signals based on their center frequency characterization.

Values

Table 380

Name	Description
uhAnoiseGen_Woofer	Corresponds to noise generator center frequency of about 70 Hz. This is mainly used to calibrate subwoofers.
uhAnoiseGen_Normal	Corresponds to noise generator center frequency of about 700 Hz. This is the most commonly used noise signal used for calibrating speakers. This is the noise type to which the noise generator is initialized; hence, it is mandatory for the platform instance to supply this noise type.
uhAnoiseGen_Tweeter	Corresponds to noise generator center frequency of about 10000 Hz. This can be used to calibrate high-frequency speakers (tweeters).

7.2.5.2.1.2 uhAnoiseGen_NoiseTypeSet_t

Signature

```
typedef UInt32 uhAnoiseGen_NoiseTypeSet_t, *puhAnoiseGen_NoiseTypeSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of noise types that are typically constructed by logical OR-ing of values of type `uhAnoiseGen_NoiseType_t`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.5.2.1.3 uhAnoiseGen_ChanIndex_t

Signature

```
typedef enum _uhAnoiseGen_ChanIndex_t {
    uhAnoiseGen_None           = 0,
    uhAnoiseGen_Left           = 1,
    uhAnoiseGen_Right          = 2,
    uhAnoiseGen_Center         = 3,
    uhAnoiseGen_Surround       = 4,
    uhAnoiseGen_LeftSurround   = 5,
    uhAnoiseGen_RightSurround  = 6,
    uhAnoiseGen_CenterSurround = 7,
    uhAnoiseGen_LeftCenter     = 8,
    uhAnoiseGen_RightCenter    = 9,
    uhAnoiseGen_Lfe            = 10
} uhAnoiseGen_ChanIndex_t, *puhAnoiseGen_ChanIndex_t;
```

Qualifiers
None.

Description
This type is used to denote the input/output (stream) audio channels. The audio channels form a superset of different prevalent channel configurations. E.g. a Dolby prologic decoded output would use the Left, Right, Center and Surround channels, a typical Dolby AC3 5.1 decoded output would use the Left, Right, Center, LeftSurround, RightSurround and LFE channels, and a typical 7.1 output would use Left, Right, Center, LeftSurround, RightSurround, LeftCenter, RightCenter and LFE channels.

Values

Table 381

Name	Description
uhAnoiseGen_None	Value used corresponding to no channel being selected.
uhAnoiseGen_Left	Left channel.
uhAnoiseGen_Right	Right channel.
uhAnoiseGen_Center	Center channel.
uhAnoiseGen_Surround	Surround channel.
uhAnoiseGen_LeftSurround	Left surround channel.
uhAnoiseGen_RightSurround	Right surround channel.
uhAnoiseGen_CenterSurround	Center surround channel.
uhAnoiseGen_LeftCenter	Left center channel.
uhAnoiseGen_RightCenter	Right center channel.
uhAnoiseGen_Lfe	Low frequency effects channel.

7.2.5.2.2 Model Types & Constants

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.5.3 Logical Component

7.2.5.3.1 Interface-Role Model

Figure 51 depicts the interface-role model of the ANG logical component. It shows the interfaces and the roles involved with this component.

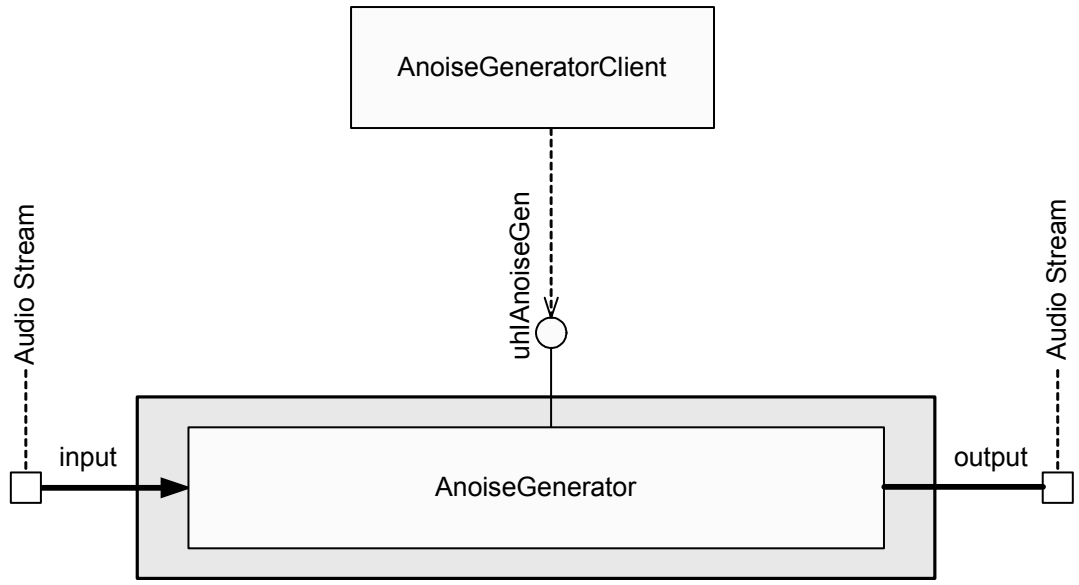


Figure 51 — Interface Role Model

The grey box in Figure 51 denotes an **Audio Noise Generator** logical component containing a single `AnoiseGenerator` role. This role provides one mandatory (solid line) control interface called `UhIAnoiseGen` to its client, the AnoiseGeneratorClient. There are no notification interfaces since the control is an open-loop one.

7.2.5.3.2 Diversity

7.2.5.3.2.1 Provided Interfaces

Table 382

Role	Interface	Presence
AnoiseGenerator	`UhIAnoiseGen`	mandatory

7.2.5.3.2.2 Configurable Items

Table 383

Role	Attribute
AnoiseGenerator	`suppNoiseTypes`

7.2.5.3.2.3 Constraints

```
suppNoiseTypes & uhAnoiseGen_Normal != 0x00000000
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.5.3.3 Instantiation

7.2.5.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 384

Type	Object	Multiplicity
AnoiseGenerator	ang	1

7.2.5.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— `ang.currentNoiseType = uhAnoiseGen_Normal`
— `ang.currentNoiseGenChan = uhAnoiseGen_None`
— `ang.generatorState = False`

7.2.5.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier “thread-safe”.

7.2.5.4 Roles

7.2.5.4.1 AnoiseGenerator

Signature

```
role AnoiseGenerator {
    const uhAnoiseGen_NoiseTypeSet_t suppNoiseTypes;
    uhAnoiseGen_NoiseType_t          currentNoiseType;
    uhAnoiseGen_ChanIndex_t          currentNoiseGenChan;
    Bool                          generatorState;
   }
```

Qualifiers
— root

Description
The AnoiseGenerator role performs the function of accepting NG settings from the user, and executing noise generation function in the specified output audio channel.

Independent Attributes

Table 385

Name	Description
`suppNoiseTypes`	The set of supported noise types for the ANG.
`currentNoiseType`	The user setting for the noise type used by the ANG.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
currentNoiseGenChan	The current channel into which noise is injected in the active (ON) state.
generatorState	Indicates whether the noise generator is activated (TRUE) or deactivated (FALSE). When the noise generator is switched on, it outputs noise signal on the channel specified in currentNoiseGenChan. A channel may be selected during the ON state. When the noise generator is switched off, the audio data presented at the input is provided at the output without modification.

Invariants
— currentNoiseType in suppNoiseTypes

Instantiation
An AnoiseGenerator is always instantiated as part of an Audio Noise Generator; see subclause 7.2.5.3.3.

Streaming Behavior
The speaker role assumes a single input stream to the LC as well as a single output stream. The consumption by the LC on the input side is discontinued when the NG functionality is activated. The noise generator does not need to be deactivated for the noise output to switch to a different channel.

Active Behavior
None.

Execution Constraints
None.

7.2.5.4.2 AnoiseGeneratorClient

Signature
```
role AnoiseGeneratorClient {}
```

Qualifiers
— actor

Description
A Noise-Generator Client represents a user of the UhIAnoiseGen interface.

Execution Constraints
None

7.2.5.5 Interfaces

7.2.5.5.1 uhlAnoiseGen

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface of an Audio Noise Generator provides functions to set and retrieve NG preferences.

Interface ID
`uuid(E206129C-62CB-458E-B69E-04A1CA5AEF97)`

7.2.5.5.1.1 NoiseGeneratorEnable

Signature
```
uhErrorCode_t NoiseGeneratorEnable ( [in] Bool state );
```

Qualifiers
— synchronous

— single-threaded

Description
Activates or deactivates the noise generator.

Parameters

Table 386

Name	Description
`state`	Indicates whether the noise generator should be activated or deactivated. If True, the noise generator is activated. If False the noise generator is deactivated.

Return Values
Standard.

Precondition
`True`

Action
— Modify `ang.generatorState`

Postcondition
`ang.generatorState == state`

7.2.5.5.1.2 GetNoiseGeneratorEnabled

Signature
```
uhErrorCode_t GetNoiseGeneratorEnabled ( [out] pBool pState );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Indicates whether noise generation is active.

Parameters

Table 387

Name	Description
`pState`	Output parameter returning the state of the ANG. A value of True indicates NG is active.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pState == ang.generatorState`

7.2.5.5.1.3 GetSuppNoiseTypes

Signature
```
uhErrorCode_t GetSuppNoiseTypes (
    [out] puhAnoiseGen_NoiseTypeSet_t pNoiseTypes );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns the set of noise types supported by the ANG.

Parameters

Table 388

Name	Description
`pNoiseTypes`	Output parameter returning the set of noise types supported by the ANG.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None

Postcondition
```
*pNoiseTypes == ang.suppNoiseTypes
```

7.2.5.5.1.4 SetNoise

Signature
```
uhErrorCode_t SetNoise (
    [in] uhAnoiseGen_NoiseType_t noiseType,
    [in] uhAnoiseGen_ChanIndex_t chan );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the type of noise signal to be generated, and the audio channel on which noise will be injected on activation of the ANG.

Parameters

Table 389

Name	Description
`noiseType`	The type of noise signal to be generated on activation of the ANG.
`chan`	The channel on which noise will be sourced in the active (ON) state.

Return Values
Standard.

Precondition
```
(noiseType & ang.suppNoiseTypes) != 0
```

Action
— Modify `ang.currentNoiseType`

— Modify `ang.currentNoiseGenChan`

Postcondition
— `ang.currentNoiseType == noiseType`
— `ang.currentNoiseGenChan == chan`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.5.5.1.5 GetNoise

Signature

```
uhErrorCode_t GetNoise (
    [out] puhAnoiseGen_NoiseType_t pNoiseType,
    [out] puhAnoiseGen_ChanIndex_t pChan );
```

Qualifiers

— synchronous

— thread-safe

Description

Retrieves the current noise type and channel setting for the ANG.

Parameters

Table 390

Name	Description
`pNoiseType`	Output parameter returning the current noise type setting for the ANG.
`pChan`	Output parameter returning the channel in which noise will be injected on activation of the ANG.

Return Values

Standard.

Precondition

`True`

Action

— None

Postcondition

— `*pNoiseType == ang.currentNoiseType`

— `*pChan == ang.currentNoiseGenChan`

7.2.6 UhlAprogSel

7.2.6.1 Concepts

7.2.6.1.1 Introduction

The Audio Program Selector logical component is designed to provide program selection control for analogue audio transmissions, analogue baseband inputs, and digital audio streams.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

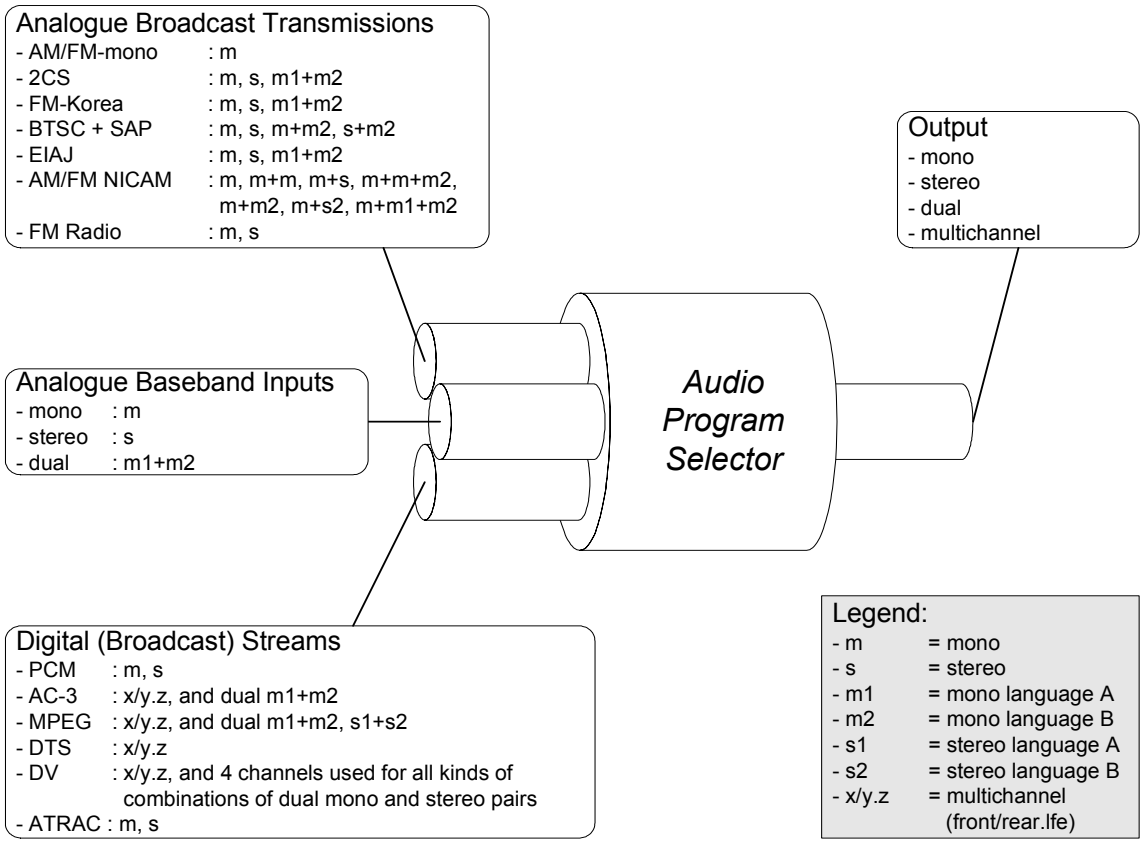


Figure 52 — Schematic overview of the Audio Program Selector.

Figure 52 depicts the Audio Program Selector with on the left hand side the possible input streams and their program capabilities and on the right hand side the capabilities of the resulting stream after selection.

7.2.6.1.2 Program Selection

In the context of the Audio Program Selector logical component, the concept of program is defined as follows:

A program is defined as a gathering (a set) of one or more audio channels with a unique audio content (often unique language) associated with it.

Consequently, an audio stream containing a mono, a stereo or a multi-channel signal (e.g. three front channels, two rear channels) constitutes a single program, while an audio stream containing a dual language signal constitutes two programs, with different languages.

For example within the AC-3 standard, in all "audio coding modes" the number of program in the decoded audio signal is one, except for the "1+1" audio coding mode where the decoded audio signal contains two programs.

A user of the program selection process is given the possibility to give a preference towards the program that should be further processed. The preference can be given among a range of program numbers supported by the platform realization.

For instance, a system supporting the DV standard may allow program choices in a range extending from the first program until the fourth program; a system supporting only AC-3 decoding, PCM and decoding of analogue broadcast transmission signals might then only allow a program choice between the first and the second program.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the program preference is not applicable for the current audio stream, because the program is not present, then a fallback scenario should determine which other program to select.

The fallback scenario is the following: if the program preference is set to the nth program and this program is not present, then the $(n-1)^{th}$ program becomes the (temporary) preferred choice, if this program is not present, the fallback scenario continues with the $(n-2)^{th}$ program and so forth until a program is finally present; see Figure 53.

When a change occurs in the number of programs in the audio stream or in the category of the audio signal, then the initial program preference is again considered. If with this inputs change the initial program preference is valid (i.e. the nth program is present), then the preferred program is directly selected, otherwise the fallback scenario starts again (i.e. the $(n-1)^{th}$ program is checked for presence), without taking into account what the temporary program preference could have been before the inputs change occurred.

For analogue audio transmission signals, it must also be possible to suppress the additional carrier containing the stereo or additional language information in case of bad (noisy) reception. In this case, the first carrier must be selectable, which is also known as a “forced mono” selection. This is realized by selecting the “Program Base” as the preferred program. The “Program Base” can only be available for analogue audio transmission signals. The fallback scenario does not apply when this program preference is selected.

For some applications (e.g. recording), it can be necessary to have also the choice of selecting the entire content of the audio stream. This is realized by selecting the “Program Full” as the preferred program; the output stream is then equal to what is applied at the input of the Audio Program Selector. The fallback scenario does not apply when this program preference is selected.

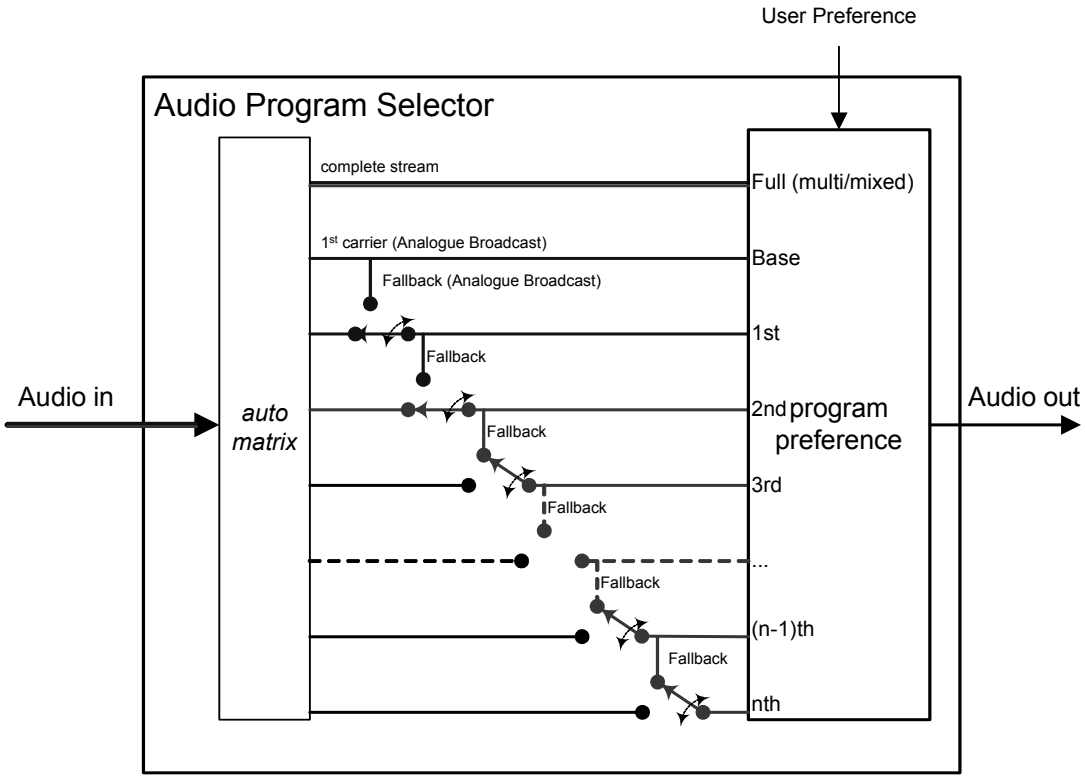


Figure 53 — Example of the structure of the Audio Program Selector: “Program Full”, “Program Base”, and “1st Program” until “nth Program” can be selected.

In the example of Figure 53, the input stream contains two programs (the 1st Program depicted in blue, the 2nd Program in red), a preference for the mth program with $m > 2$, will imply the application of the fallback scenario, resulting in the selection of the 2nd Program.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For each of the category of audio signals (i.e. signals from analogue audio transmissions, signals from analogue baseband inputs, and digital audio signals) a control is provided to the client to set a preference among the set of supported programs.

When the audio signal changes from one category to another category the program preference set for the new category is selected and applied.

The Audio Program Selector only selects programs, consequently the channel configuration of the program selected remains unchanged by the selection process: a stereo program that has been selected from a dual stereo stream stays a stereo program and is outputted in that way. The only exception is in the case of digital streams, when "Program Full" is the preferred Program and when the Output mode is set to the mixed mode; see subclause 7.2.6.1.5.1 for details.

7.2.6.1.3 Analogue broadcast transmissions

Besides a mono or stereo audio program, an analogue audio transmission can contain up to 3 different audio programs:

— FM 2CS : Language I and Language II.

— FM Korea : Language I and Language II.

— FM BTSC : Main program and Second Audio Program (SAP).

— FM EIAJ : Language I and Language II.

— AM/FM NICAM : Main program (language I), Language A and Language B.

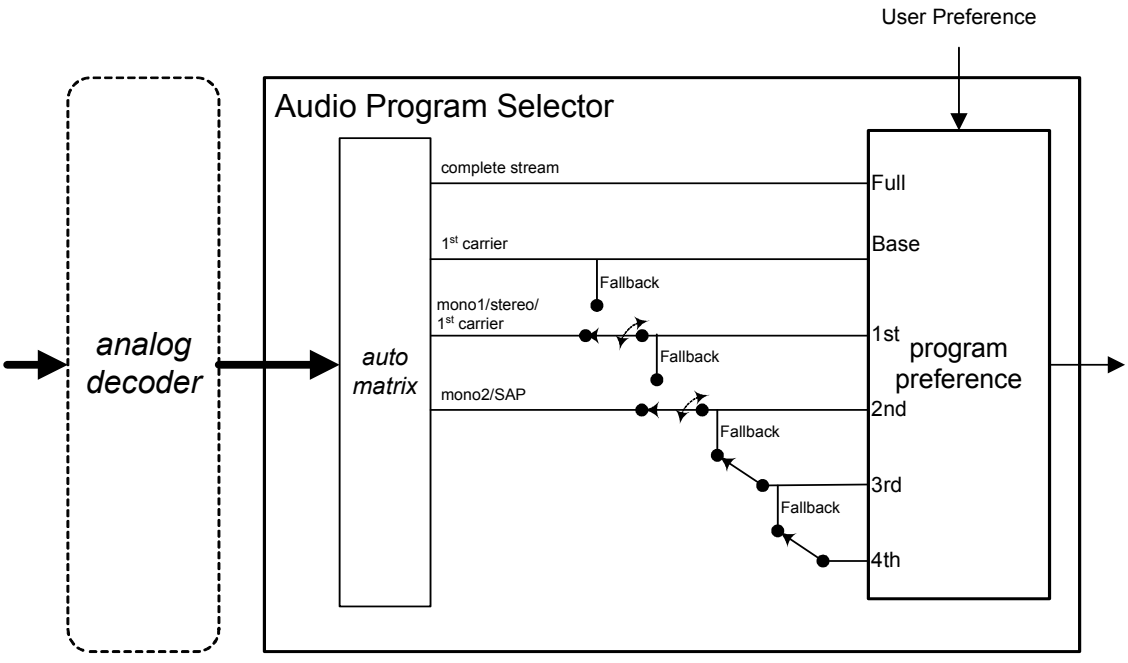


Figure 54 — General structure of the Audio Program Selector for analogue broadcast input signals.

— In Figure 54 the analog audio decoder is providing the programs from which the Program Selector can select. The Program Selector can have until six different choices representing the possible programs.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— "Program Full" is used to skip the selection of programs in the audio input stream: the output is equal to what is applied at the input.

— "Program Base" represents the "forced mono" program, which is one to one related with the 1st sound carrier of the analog audio broadcast.

— Typically "Program First" is predestined for selecting the first program in the audio stream, i.e. the main program, or the 'language A' program in case of a bilingual audio stream.

— "Program Second" is predestined for selecting the second program in the audio stream, i.e. the 'language B' program in case of a bilingual audio stream.

— For analogue audio transmission inputs, "Program Third" and "Program Fourth" always fall back at least to "Program Second", and can fall back to "Program First"; see Table 391.

— In the case of a NICAM transmission, the digital quality audio signal (i.e. the NICAM audio signal) is to be selected by giving the preference to the "Program First" (or "Program Second" if the Language B is preferred over the Language A in the case of a Bilingual audio stream). The base mono channel (1st carrier) will be selected if the preference is given to the "Program Base" (a.k.a. forced mono). The base mono channel (1st carrier) will also be selected in case the NICAM signal contains data, with the preference given to the "Program First" or the "Program Second". This is shown in Table 391.

— The selected program is set as a preference; in that way, the Audio Program Selector can select the proper program(s) to output when the input signal changes without client interaction.

— Except for the situation where a NICAM transmission changes from 'related' to 'unrelated'; in such a case the selected program may be autonomously overruled by the platform selecting the "Program Base", such that the audio output signal remains related to the video program.

Table 391 — The program selection for analogue broadcast input signals.

Input signal	Program Full	Program Base	Program First	Program Second	Program Third	Program Fourth
AM/FM Mono	Mono	Mono				
2CS Stereo	Stereo	Mono	Stereo			
2CS Bilingual	Bilingual (Dual-mono)	Mono (Language A)	Language A	Language B		
AM/FM + NICAM Mono	AM/FM + NICAM Mono	Mono (AM/FM)	NICAM Mono			
AM/FM + NICAM Stereo	AM/FM + NICAM Stereo	Mono (AM/FM)	NICAM Stereo			
AM/FM + NICAM Bilingual	AM/FM + NICAM Bilingual	Mono (AM/FM)	NICAM Language A	NICAM Language B		
AM/FM + NICAM Data	AM/FM + NICAM Data	Mono (AM/FM)				
AM/FM + NICAM unrelated Mono	AM/FM + NICAM unrelated Mono	Mono (AM/FM)	NICAM Mono			

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Input signal	Program Full	Program Base	Program First	Program Second	Program Third	Program Fourth
AM/FM + NICAM unrelated Stereo	AM/FM + NICAM unrelated Stereo	Mono (AM/FM)	NICAM Stereo			
AM/FM + NICAM unrelated Bilingual	AM/FM + NICAM unrelated Bilingual	Mono (AM/FM)	NICAM Language A	NICAM Language B		
AM/FM + NICAM unrelated Data	AM/FM + NICAM unrelated Data	Mono (AM/FM)				
BTSC Stereo	Stereo	Mono	Stereo			
BTSC Mono + SAP	Mono + SAP	Mono		SAP		
BTSC Stereo + SAP	Stereo + SAP	Mono	Stereo	SAP		
EIAJ Stereo	Stereo	Mono	Stereo			
EIAJ Bilingual	Bilingual (Dual-mono)	Mono (Language A)	Language A	Language B		
FM-Radio Mono	Mono	Mono				
FM-Radio Stereo	Stereo	Mono	Stereo			

7.2.6.1.4 Analogue baseband inputs

— In the case of a baseband input, the audio stream is considered an audio input comprising one program for the ‘left’ audio signal and one program for the ‘right’ audio signal. The common assumption is that such input is used for feeding in stereo signals only, but it could as well be dual language or mono (e.g. mono on ‘left’ or mono on ‘right’).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

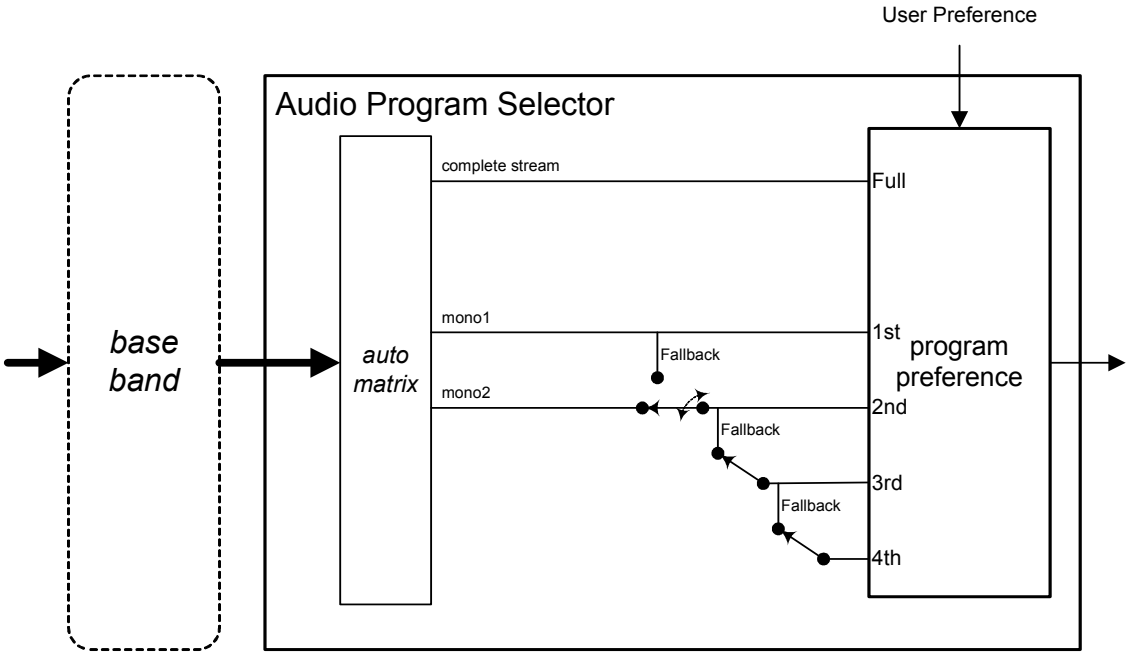


Figure 55 — General structure of the Audio Program Selector for analogue baseband input signals.

— Selecting “Program Full” the output will always be equal to what is applied at the input.

— For analogue baseband inputs, “Program Base” is not available since there is no need to select between signal quality alternatives (e.g. 1st (mono) carrier versus stereo or dual carrier).

— Selecting “Program First” will result in a mono output signal representing the main (language) program.

— Selecting “Program Second” will again represent the secondary (language) program.

— For analogue baseband inputs, “Program Third” and “Program Fourth” always fall back at least to “Program Second”, and can fall back to “Program First”; see Table 392.

Table 392 — The program selection for analogue baseband input signals.

Input signal [10]	**Program Full**	**Program Base**	**Program First**	**Program Second**	**Program Third**	**Program Fourth**
Mono	Mono		Mono			
Stereo	Stereo		Mono (Left channel)	Mono (Right channel)		
Dual-mono	Dual-mono		Mono (Left channel/ Language A)	Mono (Right channel/ Language B)		

10 Input signals mono, stereo, dual, ‘mono on left only’, and ‘mono on right only’ are considered signal properties that can not be detected reliably, but added here for completeness.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Input signal [10]	Program Full	Program Base	Program First	Program Second	Program Third	Program Fourth
Mono on left only (silence on right)	Mono on left only (silence on right)		Mono (Left channel)	Mono (silence)		
Mono on right only (silence on left)	Mono on right only (silence on right)		Mono (silence)	Mono (Right channel)		

7.2.6.1.5 Digital (broadcast) streams

Besides mono, stereo and multichannel programs a digital stream can also provide dual and up to four language programs.

— AC-3 : Language I and Language II (2x mono).

— MPEG audio : Language I and Language II (2x mono, 2x stereo, 1 mono + 1 stereo, 1 multi-channel (3/0) + 1 mono), optionally Language III (2x mono + stereo).

— DV audio : Language I and Language II, optionally Language III and Language IV (multiple combinations of mono and/or stereo).

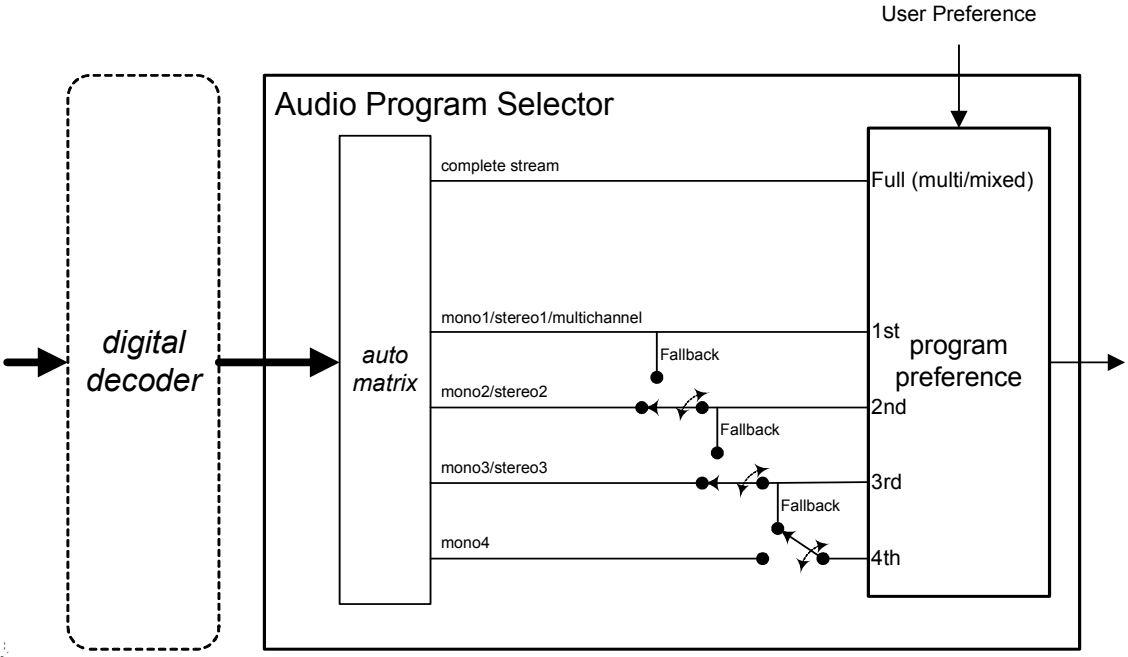


Figure 56 — General structure of the Audio Program Selector for digital (broadcast) signals.

— The program selection closely follows the partitioning as used for the analog broadcast and baseband signals.

— Selecting “Program Full” the output will contain what is applied at the input.

— For digital (broadcast) streams, “Program Base” is not available since there is no need to select between signal quality alternatives (e.g. 1st (mono) carrier versus stereo or dual carrier).

— Selecting “Program First” will result in an output signal containing the main (language) program.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Selecting “Program Second” will again result in an output signal containing the secondary (language) program.

— Selecting “Program Third” or “Program Fourth” will result in an output signal containing the third (language) program or the fourth (language) program, respectively.

— For digital streams, “Program Second”, “Program Third”, and “Program Fourth” always fall back to “Program First” in case of a mono, a stereo or a multichannel input signal; see Table 393. In those cases there is no actual effect on the input signal, which will be forwarded to the output via a loop through.

Table 393 — The program selection for digital (broadcast) signals.

Input signal	Prog Full	Prog Base	Prog First	Prog Second	Prog Third	Prog Fourth
PCM, DV-2ch Mono	Mono		Mono			
PCM, DV-2ch Stereo	Stereo		Stereo			
AC-3, MPEG, DTS, DV-4ch 1/0	Mono $_{center}$		Mono $_{center}$			
AC-3, MPEG, DTS, DV-4ch 2/0	Stereo		Stereo			
AC-3, MPEG, DTS 2/1	Multichannel (2/1)		Multichannel (2/1)			
AC-3, MPEG, DTS, DV-4ch 2/2	Multichannel (2/2)		Multichannel (2/2)			
AC-3, MPEG, DTS, DV-4ch 3/0	Multichannel (3/0)		Multichannel (3/0)			
AC-3, MPEG, DTS, DV-4ch 3/1	Multichannel (3/1)		Multichannel (3/1)			
AC-3, MPEG, DTS 3/2	Multichannel (3/2)		Multichannel (3/2)			
DTS 3/3	Multichannel (3/3)		Multichannel (3/3)			
DTS 3/4	Multichannel (3/4)		Multichannel (3/4)			
DTS 4/2	Multichannel (4/2)		Multichannel (4/2)			
DTS 4/4	Multichannel (4/4)		Multichannel (4/4)			
MPEG, DTS 5/2	Multichannel (5/2)		Multichannel (5/2)			
DTS 5/3	Multichannel (5/3)		Multichannel (5/3)			
AC-3 karaoke	Multichannel		Multichannel			

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Input signal	Prog Full	Prog Base	Prog First	Prog Second	Prog Third	Prog Fourth
AC-3, MPEG, DV-2ch 1+1	Dual-mono		Language A	Language B		
MPEG 1+2	MPEG 1+2		Mono 1 (Language A)	Stereo 2 (Language B)		
MPEG, DV-4ch 2+2	Dual-stereo		Stereo 1 (Language A)	Stereo 2 (Language B)		
DV-4ch 3+1	DV-4ch 3+1		MultiChannel 1 (3/0) (Language A)	Mono 2 (Language B)		
MPEG 3+2	MPEG 3+2		MultiChannel 1 (3/0) (Language A)	Stereo 2 (Language B)		
DV-4ch 1+1+0+0	DV-4ch 1+1+0+0		Mono 1 (Language A)	Mono 2 (Language B)		
DV-4ch 1+0+1+0	DV-4ch 1+0+1+0		Mono 1 (Language A)		Mono 3 (Language C)	
DV-4ch 1+1+1+0	DV-4ch 1+1+1+0		Mono 1 (Language A)	Mono 2 (Language B)	Mono 3 (Language C)	
DV-4ch 1+0+1+1	DV-4ch 1+0+1+1		Mono 1 (Language A)		Mono 3 (Language C)	Mono 4 (Language D)
DV-4ch 1+1+1+1	DV-4ch 1+1+1+1		Mono 1 (Language A)	Mono 2 (Language B)	Mono 3 (Language C)	Mono 4 (Language D)
DV-4ch 1+0+2	DV-4ch 1+0+2		Mono 1 (Language A)		Stereo 3 (Language C)	
MPEG, DV-4ch 1+1+2	MPEG, DV-4ch 1+1+2		Mono 1 (Language A)	Mono 2 (Language B)	Stereo 3 (Language C)	
DV-4ch 2+1+0	DV-4ch 2+1+0		Stereo 1 (Language A)	Mono 2 (Language B)		
DV-4ch 2+1+1	DV-4ch 2+1+1		Stereo 1 (Language A)	Mono 2 (Language B)	Mono 3 (Language C)	

7.2.6.1.5.1 Mixing mode

In the particular case of digital (broadcast) signals, the output can be configured to two modes when “Program Full” is selected: one mode where the programs in the audio stream remain separated and another mode where the programs are mixed into a single program.

When the programs are mixed into a single program, matching audio channels are added together, i.e. left channels are added to left channels, right channels are added to right channels, etc. The output signal will have the superset of the audio channels of the programs.

When one of the audio programs is a mono signal then three cases can be identified:

— At least one of the other audio programs is a multichannel program and contains a front center channel, then the mono signal will be added to the front center channel of the other (relevant) audio program(s);

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— None of the other audio programs contains a front center channel, but at least one program is a stereo signal, then the mono signal will be added after proper scaling (for total energy preservation) to the front left and to the front right channels of the other (relevant) audio program(s);

— All the audio programs are mono signals then all the (mono) signals are added together.

7.2.6.2 Types & Constants

7.2.6.2.1 Public Types & Constants

7.2.6.2.1.1 Error Codes

None.

7.2.6.2.1.2 uhAprogSel_Category_t

Signature

```
typedef enum _uhAprogSel_Category_t {
    uhAprogSel_CategoryInvalid      = -2,
    uhAprogSel_CategoryUnknown      = -1,
    uhAprogSel_CategoryAnaBroadcast =  0,
    uhAprogSel_CategoryAnaBaseBand  =  1,
    uhAprogSel_CategoryDigitalAudio =  2
} uhAprogSel_Category_t, *puhAprogSel_Category_t;
```

Qualifiers
None.

Description
Values of this type represent the categories of audio signal the audio input stream can belong to.

Values

Table 394

Name	Description
uhAprogSel_CategoryInvalid	The category of the audio signal is temporarily not available.
uhAprogSel_CategoryUnknown	The category of the audio signal is available, yet it could not be determined.
uhAprogSel_CategoryAnaBroadcast	The audio input stream contains a signal from analogue audio transmission origin.
uhAprogSel_CategoryAnaBaseBand	The audio input stream contains an analogue baseband signal.
uhAprogSel_CategoryDigitalAudio	The audio input stream contains a digital audio signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.6.2.1.3 uhAprogSel_Program_t

Signature

```
typedef enum _uhAprogSel_Program_t {
    uhAprogSel_ProgInvalid    = 0x80000000,
    uhAprogSel_ProgUnknown    = 0x40000000,
    uhAprogSel_ProgFull       = 0x00000001,
    uhAprogSel_ProgBase       = 0x00000002,
    uhAprogSel_ProgFirst      = 0x00000004,
    uhAprogSel_ProgSecond     = 0x00000008,
    uhAprogSel_ProgThird      = 0x00000010,
    uhAprogSel_ProgFourth     = 0x00000020
} uhAprogSel_Program_t, *puhAprogSel_Program_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the selectable programs from analogue broadcast, baseband input signals, and digital (broadcast) input streams.

Values

Table 395

Name	Description
uhAprogSel_ProgInvalid	The information over the selected program is temporarily not available.
uhAprogSel_ProgUnknown	The information over the selected program is available, yet it could not be determined.
uhAprogSel_ProgFull	No program selection is performed: the input signal is forwarded to the output via a loop through.
uhAprogSel_ProgBase	"Forced mono" selection: the 1st sound carrier of the analog audio broadcast is selected. This selection is only available for analog audio broadcast inputs.
uhAprogSel_ProgFirst	First stream (main language/"stereo" if available) preferred.
uhAprogSel_ProgSecond	Second stream (second language) preferred.
uhAprogSel_ProgThird	Third stream (third language) preferred.
uhAprogSel_ProgFourth	Fourth stream (fourth language) preferred.

Remarks

— For a conceptual description please refer to the concept subclause 7.2.6.1.2.

7.2.6.2.1.4 uhAprogSel_ProgramSet_t

Signature

```
typedef Int32 uhAprogSel_ProgramSet_t, *puhAprogSel_ProgramSet_t;
```

Qualifiers

— enum-set

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of programs.

Constraints
None.

7.2.6.2.1.5 uhAprogSel_OutputMode_t

Signature
```
typedef enum _uhAprogSel_OutputMode_t {
    uhAprogSel_OutputModeMulti = 0,
    uhAprogSel_OutputModeMixed = 1
} uhAprogSel_OutputMode_t, *puhAprogSel_OutputMode_t;
```

Qualifiers
None.

Description
Values of this type represent the output operation modes that can be used when “Program Full” is selected in case of audio input signals belonging to the digital (broadcast) signals category.

Values

Table 396

Name	Description
uhAprogSel_OutputModeMulti	The programs in the audio output stream remain separated, as they were in the audio input stream.
uhAprogSel_OutputModeMixed	The output stream contains a single program that is the result of the mixing of the audio programs contained in the input stream.

Remarks
— For a conceptual description please refer to the Concept section.

7.2.6.2.1.6 uhAprogSel_Quality_t

Signature
```
typedef enum _uhAprogSel_Quality_t {
    uhAprogSel_QualityInvalid       = -2,
    uhAprogSel_QualityUnknown       = -1,
    uhAprogSel_QualityNormal        =  0,
    uhAprogSel_QualityDual          =  1,
    uhAprogSel_QualitySurround      =  2,
    uhAprogSel_QualityMultiChannel  =  3,
    uhAprogSel_QualityMultiLanguage =  4,
    uhAprogSel_QualityKaraoke       =  5
} uhAprogSel_Quality_t, *puhAprogSel_Quality_t;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent the quality of audio streams with respect to the elementary particularities and capabilities of an audio stream.

Values

Table 397

Name	Description
uhAprogSel_QualityInvalid	The quality of the audio stream is temporarily not available.
uhAprogSel_QualityUnknown	The quality of the audio stream is available, yet it could not be determined.
uhAprogSel_QualityNormal	The signal is a standard mono or stereo signal without additional capabilities.
uhAprogSel_QualityDual	The signal provides two separate languages.
uhAprogSel_QualitySurround	The signal is a Dolby Surround or Dolby Pro-Logic encoded stereo signal.
uhAprogSel_QualityMultiChannel	The signal provides separate audio channels containing the individual front and rear audio channels.
uhAprogSel_QualityMultiLanguage	The signal provides separate audio channels containing three or more languages.
uhAprogSel_QualityKaraoke	The signal provides separate audio channels containing the instrument and voice programs of the singalong music.

7.2.6.2.1.7 uhAprogSel_Standard_t

Signature
```
typedef enum _uhAprogSel_Standard_t {
    uhAprogSel_StandardInvalid    = -2,
    uhAprogSel_StandardUnknown    = -1,
    uhAprogSel_StandardAmMono     =  0,
    uhAprogSel_StandardFmMono     =  1,
    uhAprogSel_StandardFm2Cs      =  2,
    uhAprogSel_StandardFmKorea    =  3,
    uhAprogSel_StandardAmNicam    =  4,
    uhAprogSel_StandardFmNicam    =  5,
    uhAprogSel_StandardBtsc       =  6,
    uhAprogSel_StandardEiaj       =  7,
    uhAprogSel_StandardFmRadio    =  8,
    uhAprogSel_StandardLrBaseBand =  9,
    uhAprogSel_StandardAc3        = 10,
    uhAprogSel_StandardMpeg       = 11,
    uhAprogSel_StandardDts        = 12,
    uhAprogSel_StandardDv2ch      = 13,
    uhAprogSel_StandardDv4ch      = 14,
    uhAprogSel_StandardAtrac      = 15,
    uhAprogSel_StandardPcm        = 16
} uhAprogSel_Standard_t, *puhAprogSel_Standard_t;
```

Qualifiers
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The values of this type represent audio signal standards.

Values

Table 398

Name	Description
uhAprogSel_StandardInvalid	The standard of the audio signal is temporarily not available.
uhAprogSel_StandardUnknown	The standard of the audio signal is available, yet it could not be determined.
uhAprogSel_StandardAmMono	The signal is analogue amplitude modulated (AM) mono.
uhAprogSel_StandardFmMono	The signal is analogue frequency modulated (FM) mono.
uhAprogSel_StandardFm2Cs	The signal is analogue frequency modulated 2-Carrier Sound (2CS) also known as German Stereo, with stereo or dual language capabilities.
uhAprogSel_StandardFmKorea	The signal is analogue frequency modulated 2-carrier sound also known as Korean Stereo, with stereo and dual language capabilities.
uhAprogSel_StandardAmNicam	The signal is a combination of an analogue amplitude modulated (AM) mono with additional NICAM signal, providing digital mono, stereo and dual language capabilities.
uhAprogSel_StandardFmNicam	The signal is a combination of an analogue frequency modulated (FM) mono with additional NICAM signal, providing digital mono, stereo and dual language capabilities.
uhAprogSel_StandardBtsc	The signal is analogue frequency modulated (FM) with mono and stereo capabilities and optionally provided SAP mono.
uhAprogSel_StandardEiaj	The signal is analogue frequency modulated (FM) also known as Japan Stereo with mono, stereo and dual capabilities.
uhAprogSel_StandardFmRadio	The signal an analogue frequency modulated (FM) radio with mono or stereo capabilities.
uhAprogSel_StandardLrBaseBand	The signal is analogue baseband (left/right) with mono, stereo and dual capabilities.
uhAprogSel_StandardAc3	The signal is digital encoded AC-3 (Dolby Digital) multichannel.
uhAprogSel_StandardMpeg	The signal is digital encoded MPEG multichannel.
uhAprogSel_StandardDts	The signal is digital encoded DTS multichannel.
uhAprogSel_StandardDv2ch	The signal is digital encoded DV SD 2-channel audio with mono, stereo and dual capabilities.
uhAprogSel_StandardDv4ch	The signal is digital encoded DV SD multichannel.
uhAprogSel_StandardAtrac	The signal is digital encoded ATRAC signal.
uhAprogSel_StandardPcm	The signal is digital PCM signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.6.2.1.8 uhAprogSel_Config_t

Signature

```
typedef enum _uhAprogSel_Config_t {
    uhAprogSel_ConfigInvalid = -2,
    uhAprogSel_ConfigUnknown = -1,
    uhAprogSel_Config1s0     =  0,
    uhAprogSel_Config2s0     =  1,
    uhAprogSel_Config2s1     =  2,
    uhAprogSel_Config2s2     =  3,
    uhAprogSel_Config3s0     =  4,
    uhAprogSel_Config3s1     =  5,
    uhAprogSel_Config3s2     =  6,
    uhAprogSel_Config3s3     =  7,
    uhAprogSel_Config3s4     =  8,
    uhAprogSel_Config4s2     =  9,
    uhAprogSel_Config4s4     = 10,
    uhAprogSel_Config5s2     = 11,
    uhAprogSel_Config5s3     = 12,
    uhAprogSel_Config1p1     = 13,
    uhAprogSel_Config1p2     = 14,
    uhAprogSel_Config2p2     = 15,
    uhAprogSel_Config3p1     = 16,
    uhAprogSel_Config3p2     = 17,
    uhAprogSel_Config1p1p0p0 = 18,
    uhAprogSel_Config1p0p1p0 = 19,
    uhAprogSel_Config1p1p1p0 = 20,
    uhAprogSel_Config1p0p1p1 = 21,
    uhAprogSel_Config1p1p1p1 = 22,
    uhAprogSel_Config1p0p2   = 23,
    uhAprogSel_Config1p1p2   = 24,
    uhAprogSel_Config2p1p0   = 25,
    uhAprogSel_Config2p1p1   = 26
} uhAprogSel_Config_t, *puhAprogSel_Config_t;
```

Qualifiers
None.

Description
Values of this type represent channel configurations of audio streams.

Values

Table 399

Name	Description
uhAprogSel_ConfigInvalid	The channel configuration of the audio stream is temporarily not available.
uhAprogSel_ConfigUnknown	The channel configuration of the audio stream is available, yet it could not be determined.
uhAprogSel_Config1s0	The signal is encoded as 1/0 multichannel, providing one front (center) signal.
uhAprogSel_Config2s0	The signal is encoded as 2/0 multichannel, providing two front (left/right) signals.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhAprogSel_Config2s1	The signal is encoded as 2/1 multichannel, providing two front (left/right) and one rear (surround) signals.
uhAprogSel_Config2s2	The signal is encoded as 2/2 multichannel, providing two front (left/right) and two rear (surround left/right) signals.
uhAprogSel_Config3s0	The signal is encoded as 3/0 multichannel, providing three front (left/center/right) signals.
uhAprogSel_Config3s1	The signal is encoded as 3/1 multichannel, providing three front (left/center/right) and one rear (surround) signals.
uhAprogSel_Config3s2	The signal is encoded as 3/2 multichannel, providing three front (left/center/right) and two rear (surround left/right) signals.
uhAprogSel_Config3s3	The signal is encoded as 3/3 multichannel, providing three front (left/center/right), and three rear (surround left/center/right) signals.
uhAprogSel_Config3s4	The signal is encoded as 3/4 multichannel, providing three front (left/center/right), and four rear (surround left/left-back/right-back/ surround right) signals.
uhAprogSel_Config4s2	The signal is encoded as 4/2 multichannel, providing four front (left/left-center/right-center/right), and two rear (surround left/right) signals.
uhAprogSel_Config4s4	The signal is encoded as 4/4 multichannel, providing four front (left/left-center/right-center/right), and four rear (first and second surround left/ first and second surround right) signals.
uhAprogSel_Config5s2	The signal is encoded as 5/2 multichannel, providing three front (left/center/right), two side (center left/right) and two rear (surround left/right) signals.
uhAprogSel_Config5s3	The signal is encoded as 5/3 multichannel, providing three front (left/center/right), two side (center left/right) and three rear (surround left/center/right) signals.
uhAprogSel_Config1p1	The signal is encoded as 1+1 multichannel, providing two mono programs (language A/B).
uhAprogSel_Config1p2	The signal is encoded as 1+2 multichannel, providing one mono and one stereo program (language A/B).
uhAprogSel_Config2p2	The signal is encoded as 2+2 multichannel, providing two stereo programs (language A/B).
uhAprogSel_Config3p1	The signal is encoded as 3+1 multichannel, providing two programs. First program provides three front (left/center/right) stereo (language A) signals and second program is mono (language B).
uhAprogSel_Config3p2	The signal is encoded as 3+2 multichannel, providing two programs. First program provides three front (left/center/right) stereo (language A) signals and second program is stereo (language B).
uhAprogSel_Config1p1p0p0	The signal is encoded as 1+1+0+0 multichannel, providing two mono programs (language A/B).
uhAprogSel_Config1p0p1p0	The signal is encoded as 1+0+1+0 multichannel, providing two mono programs (language A/C).
uhAprogSel_Config1p1p1p0	The signal is encoded as 1+1+1+0 multichannel, providing three mono programs (language A/B/C).
uhAprogSel_Config1p0p1p1	The signal is encoded as 1+0+1+1 multichannel, providing three mono programs (language A/C/D).
uhAprogSel_Config1p1p1p1	The signal is encoded as 1+1+1+1 multichannel, providing four mono programs (language A/B/C/D).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhAprogSel_Config1p0p2	The signal is encoded as 1+0+2 multichannel, providing three programs. First program provides one mono (language A) signal, second program is not in use and third program is stereo (language C).
uhAprogSel_Config1p1p2	The signal is encoded as 1+1+2 multichannel, providing three programs. First and second program provide a mono (language A/B) signal, third program is stereo (language C).
uhAprogSel_Config2p1p0	The signal is encoded as 2+1+0 multichannel, providing three programs. First program provides a stereo (language A) signal, second a mono (language C) signal and the third program is not in use.
uhAprogSel_Config2p1p1	The signal is encoded as 2+1+1 multichannel, providing three programs. First program provides a stereo (language A) signal, followed by two programs with two separate mono (language B/C) signals.

7.2.6.2.1.9 uhAprogSel_Language_t

Signature

```
typedef enum _uhAprogSel_Language_t {
    uhAprogSel_LanguageInvalid  = -2,
    uhAprogSel_LanguageUnknown  = -1,
    uhAprogSel_LanguageNone     =  0,
    uhAprogSel_LanguageA        =  1,
    uhAprogSel_LanguageB        =  2,
    uhAprogSel_LanguageC        =  3,
    uhAprogSel_LanguageD        =  4,
    uhAprogSel_LanguageAB       =  5,
    uhAprogSel_LanguageAC       =  6,
    uhAprogSel_LanguageABC      =  7,
    uhAprogSel_LanguageACD      =  8,
    uhAprogSel_LanguageABCD     =  9
} uhAprogSel_Language_t, *puhAprogSel_Language_t;
```

Qualifiers

None.

Description

Values of this type specify the content of the audio output streams.

Values

Table 400

Name	Description
uhAprogSel_LanguageInvalid	The content of the audio output stream is temporarily not available.
uhAprogSel_LanguageUnknown	The content of the audio output stream is available, yet it could not be determined.
uhAprogSel_LanguageNone	The output is not a specific language signal.
uhAprogSel_LanguageA	The output represents the language A signal.
uhAprogSel_LanguageB	The output represents the language B signal.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhAprogSel_LanguageC	The output represents the language C signal.
uhAprogSel_LanguageD	The output represents the language D signal.
uhAprogSel_LanguageAB	The output represents the language A and B signals.
uhAprogSel_LanguageAC	The output represents the language A and C signals.
uhAprogSel_LanguageABC	The output represents the language A, B and C signals.
uhAprogSel_LanguageACD	The output represents the language A, C and D signals.
uhAprogSel_LanguageABCD	The output represents the language A, B, C and D signals.

7.2.6.2.1.10 uhAprogSel_InputSignal_t

Signature

```
typedef struct _uhAprogSel_InputSignal_t {
    uhAprogSel_Category_t   category;

uhAprogSel_Standard_t   standard;
    uhAprogSel_Quality_t    quality;

uhAprogSel_Config_t     config;
} uhAprogSel_InputSignal_t, *puhAprogSel_InputSignal_t;
```

Qualifiers
None.

Description
The fields in this structure represent the input stream properties.

Fields

Table 401

Name	Description
category	The category of audio signal the audio input stream is belonging to.
standard	The (original) standard of the audio input stream.
quality	The elementary quality (particularities and capabilities) of the audio input stream.
config	The channel configuration of the audio input stream.

7.2.6.2.1.11 uhAprogSel_OutputSignal_t

Signature

```
typedef struct _uhAprogSel_OutputSignal_t {
    uhAprogSel_Quality_t     quality;

uhAprogSel_Language_t    language;

uhAprogSel_Config_t      config;
} uhAprogSel_OutputSignal_t, *puhAprogSel_OutputSignal_t;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The fields in this structure represent the output signal properties.

Fields

Table 402

Name	Description
`quality`	The elementary quality (particularities and capabilities) of the audio output stream.
`language`	The language (if any) of the audio output stream.
`config`	The channel configuration of the audio output stream.

7.2.6.2.1.12 uhAprogSel_Ntf_t

Signature
```
typedef enum _uhAprogSel_Ntf_t {
    uhAprogSel_SubscriptionChanged     = 0x00000001,
    uhAprogSel_InputSignalChanged      = 0x00000002,
    uhAprogSel_OutputSignalChanged     = 0x00000004,
    uhAprogSel_ActualProgramChanged    = 0x00000008
} uhAprogSel_Ntf_t, *puhAprogSel_Ntf_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAprogSelNtf`.

Values

Table 403

Name	Description
`uhAprogSel_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhAprogSel_InputSignalChanged`	Represents `OnInputSignalChanged`.
`uhAprogSel_OutputSignalChanged`	Represents `OnOutputSignalChanged`.
`uhAprogSel_ActualProgramChanged`	Represents `OnActualProgramChanged`.

7.2.6.2.1.13 uhAprogSel_NtfSet_t

Signature
```
typedef Int32 uhAprogSel_NtfSet_t, *puhAprogSel_NtfSet_t;
```

Qualifiers
— enum-set

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type `uhAprogSel_Ntf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints
None.

7.2.6.2.1.14 uhAprogSel_AllNtfs

Signature
```
const uhAprogSel_NtfSet_t uhAprogSel_AllNtfs = 0x0000000F;
```

Qualifiers
None.

Description
Defines the set of all values of type uhAprogSel_Ntf_t. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

7.2.6.2.2 Model Types & Constants

None.

7.2.6.3 Logical Component

7.2.6.3.1 Interface-Role Model

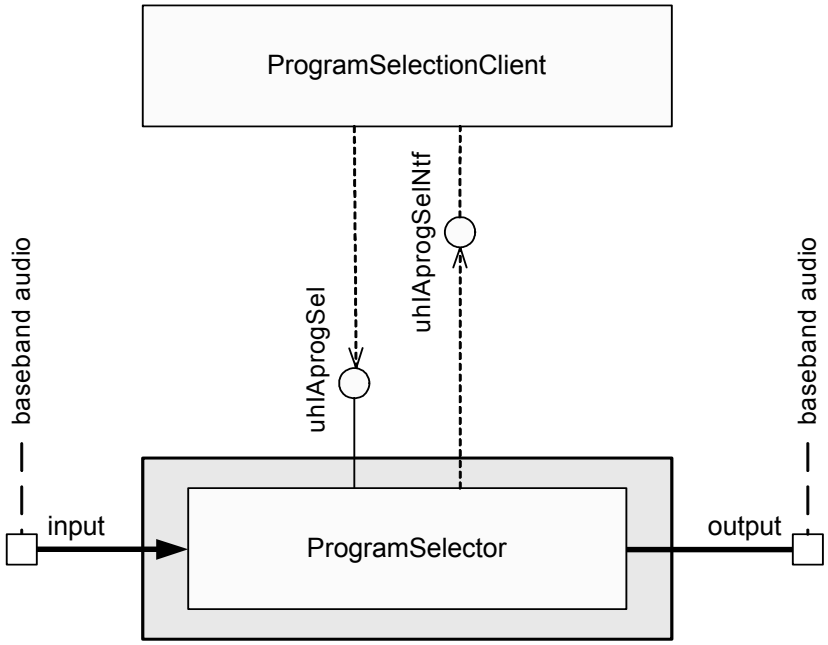


Figure 57 — Interface Role Model

The grey box in Figure 57 denotes an Audio Program Selector logical component containing a single `ProgramSelector` role.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
467
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This role provides one mandatory (solid line) control interface (`UhIAprogSel`).

A client can optionally provide a call-back (dotted lines) interface `uhIAprogSelNtf`. This allows the client to receive notifications of events in the Audio Program Selector.

7.2.6.3.2 Diversity

7.2.6.3.2.1 Provided Interfaces

Table 404

Role	Interface	Presence
`ProgramSelector`	`UhIAprogSel`	mandatory
`ProgramSelectionClient`	`uhIAprogSelNtf`	optional

7.2.6.3.2.2 Configurable Items

Table 405

Role	Attribute
`ProgramSelector`	`suppCategories`
	`suppPrograms`
	`suppOutputModes`
	`isNicamFallbackEnabled`

7.2.6.3.2.3 Constraints

```
— !(suppCategories[uhAprogSel_CategoryInvalid])
— !(suppCategories[uhAprogSel_CategoryUnknown])
— For each c in uhAprogSel_Category_t
   — (suppPrograms[c] & uhAprogSel_ProgFull)
   — !(suppPrograms[c] & uhAprogSel_ProgInvalid)
   — !(suppPrograms[c] & uhAprogSel_ProgUnknown)
   — !(suppPrograms[uhAprogSel_CategoryAnaBaseBand] & uhAprogSel_ProgBase)
   — !(suppPrograms[uhAprogSel_CategoryDigitalAudio] & uhAprogSel_ProgBase)
   — (suppOutputModes[uhAprogSel_OutputModeMulti]                              ||
     suppOutputModes[uhAprogSel_OutputModeMixed])
— If !(suppCategories[uhAprogSel_CategoryAnaBroadcast])
   — isNicamFallbackEnabled == False
```

7.2.6.3.3 Instantiation

7.2.6.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 406

Type	Object	Multiplicity
ProgramSelector	aps	1

7.2.6.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— aps.inputSignal.category == uhAprogSel_CategoryInvalid
— aps.inputSignal.standard == uhAprogSel_StandardInvalid
— aps.inputSignal.quality == uhAprogSel_QualityInvalid
— aps.inputSignal.config == uhAprogSel_ConfigInvalid
— aps.outputSignal.quality == uhAprogSel_QualityInvalid
— aps.outputSignal.language == uhAprogSel_LanguageInvalid
— aps.outputSignal.config == uhAprogSel_ConfigInvalid
— For each i in uhAprogSel_Category_t
  — aps.prefProgram[i] == uhAprogSel_ProgFull
  — aps.currentProgram == uhAprogSel_ProgInvalid
— If suppOutputModes[uhAprogSel_OutputModeMulti]
  — aps.outputMode == uhAprogSel_OutputModeMulti
else
  — aps.outputMode == uhAprogSel_OutputModeMixed

7.2.6.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

7.2.6.4 Roles

7.2.6.4.1 ProgramSelector

Signature

```
role ProgramSelector {
    uhAprogSel_InputSignal_t         inputSignal;
uhAprogSel_OutputSignal_t        outputSignal;
    const Bool                       suppCategories[uhAprogSel_Category_t];
    const uhAprogSel_ProgramSet_t    suppPrograms[uhAprogSel_Category_t];
    uhAprogSel_Program_t             prefProgram[uhAprogSel_Category_t];
    uhAprogSel_Program_t             currentProgram;
    const Bool                       suppOutputModes[uhAprogSel_OutputMode_t];
    uhAprogSel_OutputMode_t          outputMode;
    const Bool                       isNicamFallbackEnabled;
}
```

Qualifiers

— root

Description

A Program Selector selects, according to the user preference, the program that it should stream out among the audio programs present in its audio input stream. A Program Selector provides functionality to the client(s)

-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

to select the program to be streamed out for each category of audio input streams: signals from analogue audio transmissions, signals from analogue baseband inputs, and digital audio signals.

A Program Selector provides notifications of changes in the program that is selected and hence streamed out.

Independent Attributes

Table 407

Name	Description
`inputSignal`	The properties of the audio input stream.
`outputSignal`	The properties of the audio output stream.
`suppCategories`	Array indicating for each category of audio signals whether the setting of a preference for a program is supported by the Program Selector.
`suppPrograms`	For each category of audio signals, the set of programs for which selection is supported by the Program Selector.
`prefProgram`	For each category of audio signals, the program that should preferably be selected whenever an audio input stream of this category is present.
`currentProgram`	The program that is actually selected for the audio input stream. The actual program selected may not be equal to the preferred program for the current audio input stream: if the preferred program is not present, a fallback program is selected; see subclause 7.2.6.1.2 for a complete description of the fallback scenario.
`suppOutputModes`	Array indicating for each output operation mode whether this output operation mode is supported by the Program Selector.
`outputMode`	Indicates the output operation mode that should be used when “Program Full” is selected in the case of an audio input signal belonging to the digital (broadcast) signals category.
`isNicamFallbackEnabled`	Indicates whether the platform will or not autonomously overrule the preferred program with the “Program Base” in the case that the NICAM signal changes from 'related' to 'unrelated' NICAM. This indicates that the analogue mono (the still 'related' part) is preferred if this situation occurs. It is `True` when the platform will autonomously overrule the preferred program; it is `False` when the platform will not autonomously overrule the preferred program.

Dependent Attributes
None.

Invariants

- For each `c` in `uhAprogSel_Category_t`
  - `prefProgram[c] in suppPrograms[c]`
  - `currentProgram in (suppPrograms[inputSignal.category] | uhAprogSel_ProgInvalid | uhAprogSel_ProgUnknown)`
- If `!(suppOutputModes[uhAprogSel_OutputModeMulti])`
  - `outputMode == uhAprogSel_OutputModeMixed`
- If `!(suppOutputModes[uhAprogSel_OutputModeMixed])`
  - `outputMode == uhAprogSel_OutputModeMulti`

Instantiation
A Program Selector is always instantiated as part of an Audio Program Selector; see subclause 7.2.6.3.3.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Streaming Behavior
The Program Selector selects among the audio program(s) that are present in its audio input stream the `currentProgram` program and streams only this `currentProgram` program out.

In general, the `currentProgram` program corresponds to the preference set in `prefProgram[inputSignal.category]` for the current audio input stream with `inputSignal.category` being the category to which the current audio input signal belongs. However the actual program selected may not be equal to the preferred program for the current audio input stream: if the preferred program is not present, a fallback program is selected; see subclause 7.2.6.1.2 for a complete description of the fallback scenario.

The actual program selected may also not be equal to the preferred program in case of a NICAM transmission with a change from 'related' to 'unrelated'. If `isNicamFallbackEnabled` is `True`, in such a case the selected program is autonomously overruled by the platform by selecting the "Program Base", such that the audio output signal remains related to the video program, otherwise the preferred program in `prefProgram[uhAprogSel_CategoryAnaBroadcast]` is selected.

Active Behavior
The Program Selector has standard notification behavior (see `uhIAprogSelNtf`).

Execution Constraints
None.

7.2.6.4.2 ProgramSelectionClient

Signature
`role` `ProgramSelectionClient {}`

Qualifiers
— actor

Description
A Program Selection Client represents a user of the `UhIAprogSel` interface.

Execution Constraints
None.

7.2.6.5 Interfaces

7.2.6.5.1 uhIAprogSel

Qualifiers
None.

Description
This interface of the Audio Program Selector provides functionality to select a certain audio program (e.g. languages) from any analogue or digital audio stream. Next to that it also provides information about the input signal (before the program selection) and the output signal (after the program selection is performed).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 3292C759-A7F5-418E-99EE-9FA5F44B0BBC )`

7.2.6.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAprogSelNtf      *pINotify,
    [in] UInt32              cookie,
    [in] uhAprogSel_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.2.6.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAprogSelNtf      *pINotify,
    [in] UInt32              cookie,
    [in] uhAprogSel_NtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

7.2.6.5.1.3 IsCategorySupported

Signature
```
uhErrorCode_t IsCategorySupported (
    [in] uhAprogSel_Category_t category,
    [out] pBool pSupported );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether setting a preference for a program is supported for the specified category of audio signals.

Parameters

Table 408

Name	Description
`category`	The category of audio signals.
`pSupported`	Output parameter returning whether setting a preference for a program is supported for the specified category of audio sgnals in `category`.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
— `category != uhAprogSel_CategoryInvalid`
— `category != uhAprogSel_CategoryUnknown`

Action
None.

Postcondition
`*pSupported == aps.suppCategories[category]`

7.2.6.5.1.4 GetNicamFbEnabled

Signature
```
uhErrorCode_t GetNicamFbEnabled ( [out] pBool pNicamFallbackEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the platform will or not autonomously overrule the preferred program with the "Program Base" in the case that the NICAM signal changes from 'related' to 'unrelated' NICAM. This indicates that the analogue mono (the still 'related' part) is preferred if this situation occurs.

Parameters

Table 409

Name	Description
`pNicamFallbackEnabled`	Output parameter returning whether the platform will or not autonomously overrule the preferred program with the "Program Base" in the case that the NICAM signal changes from 'related' to 'unrelated' NICAM. It is `True` when the platform will autonomously overrule the preferred program; it is `False` when the platform will not autonomously overrule the preferred program.

Return Values
Standard.

Precondition
`aps.suppCategories[uhAprogSel_CategoryAnaBroadcast]`

Action
None.

Postcondition
`*pNicamFallbackEnabled == aps.isNicamFallbackEnabled`

`Remarks`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— When the platform does not overrule the preferred program in the case that the NICAM signal changes from 'related' to 'unrelated' NICAM (i.e. when `*pNicamFallbackEnabled` is `False`), it is the responsibility of the client to select the appropriate program, if this is necessary.

7.2.6.5.1.5 GetInputSignal

Signature
```
uhErrorCode_t GetInputSignal ( [out] puhAprogSel_InputSignal_t pSignal );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the signal properties of the audio input stream.

Parameters

Table 410

Name	Description
`pSignal`	Output parameter returning the properties of the audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pSignal == aps.inputSignal`

7.2.6.5.1.6 GetOutputSignal

Signature
```
uhErrorCode_t GetOutputSignal ( [out] puhAprogSel_OutputSignal_t pSignal );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the signal properties of the audio output stream.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 411

Name	Description
pSignal	Output parameter returning the properties of the audio output stream.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pSignal == aps.outputSignal

7.2.6.5.1.7 GetSuppPrograms

Signature

```
uhErrorCode_t GetSuppPrograms (
    [in] uhAprogSel_Category_t inputCategory,
    [out] puhAprogSel_ProgramSet_t pSupportedPrograms );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns the set of programs for which selection is supported by the Program Selector for the specified category of audio signals.

Parameters

Table 412

Name	Description
inputCategory	The category of audio signals for which the set of programs for which selection is supported is requested.
pSupportedPrograms	Output parameter returning for the set of programs for which selection is supported by the Program Selector.

Return Values
Standard.

Precondition
aps.suppCategories[inputCategory]

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pSupportedPrograms == aps.suppPrograms[inputCategory]
```

7.2.6.5.1.8 SetProgram

Signature
```
uhErrorCode_t SetProgram (
    [in] uhAprogSel_Category_t inputCategory,
    [in] uhAprogSel_Program_t program );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the program that should preferably be selected for the specified category of audio signals.

Parameters

Table 413

Name	Description
`inputCategory`	The category of audio signals for which the program to be selected is set.
`program`	The program that should be preferably selected for the category of audio signals `inputCategory`.

Return Values
Standard.

Precondition
— `aps.suppCategories[inputCategory]`
— `program in aps.suppPrograms[inputCategory]`

Action
— Modify `aps.prefProgram[inputCategory]`

Postcondition
`aps.prefProgram[inputCategory] == program`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.6.5.1.9 GetProgram

Signature
```
uhErrorCode_t GetProgram (
    [in] uhAprogSel_Category_t inputCategory,
    [out] puhAprogSel_Program_t pProgram );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the program that should be preferably selected whenever an audio input stream of the specified category of audio signals is present.

Parameters

Table 414

Name	Description
`inputCategory`	The category of audio signal for which the program that should be preferably used is requested.
`pProgram`	Output parameter returning the program that should be preferably selected whenever an audio stream of the specified category of audio signals `inputCategory` is present.

Return Values
Standard.

Precondition
```
aps.suppCategories[inputCategory]
```

Action
None.

Postcondition
```
*pProgram == aps.prefProgram[inputCategory]
```

7.2.6.5.1.10 GetActualProgram

Signature
```
uhErrorCode_t GetActualProgram ( [out] puhAprogSel_Program_t pActualProgram );
```

Qualifiers
— synchronous

— thread-safe

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the actual program selected by the Program Selector for the audio input stream. The actual program selected may not be equal to the preferred program for the current audio input stream: if the preferred program is not present, a fallback program is selected; see subclause 7.2.6.1.2 for a complete description of the fallback scenario.

Parameters

Table 415

Name	Description
`pActualProgram`	Output parameter returning the actual program selected for the audio input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pActualProgram == aps.currentProgram`

7.2.6.5.1.11 IsOutputModeSupported

Signature
```
uhErrorCode_t IsOutputModeSupported (
    [in] uhAprogSel_OutputMode_t outputMode,
    [out] pBool pSupported);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the specified output mode is supported.

Parameters

Table 416

Name	Description
`outputMode`	The output mode for which the support is requested.
`pSupported`	Output parameter returning whether the outputmode `outputMode` is supported.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`aps.suppCategories[uhAprogSel_CategoryDigitalAudio]`

Action
None.

Postcondition
`*pSupported == aps.suppOutputModes[outputMode]`

7.2.6.5.1.12 SetOutputMode

Signature
`uhErrorCode_t SetOutputMode ( [in] uhAprogSel_OutputMode_t outputModeSelected );`

Qualifiers
— synchronous

— single-threaded

Description
Sets the output mode that should be used when “Program Full” is selected in case of audio input signals belonging to the digital (broadcast) signals category.

Parameters

Table 417

Name	Description
`outputModeSelected`	The output operation mode that should be used when “Program Full” is selected in the case of an audio input signal belonging to the digital (broadcast) signals category.

Return Values
Standard.

Precondition
— `aps.suppCategories[uhAprogSel_CategoryDigitalAudio]`
— `aps.suppOutputModes[outputModeSelected]`

Action
— Modify `aps.outputMode`

Postcondition
`aps.outputMode == outputModeSelected`

© ISO/IEC 2007 – All rights reserved
--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.6.5.1.13 GetOutputMode

Signature
```
uhErrorCode_t GetOutputMode ( [out] puhAprogSel_OutputMode_t pOutputMode );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the output mode that should be used when “Program Full” is selected in case of audio input signals belonging to the digital (broadcast) signals category.

Parameters

Table 418

Name	Description
`pOutputMode`	Output parameter returning the output operation mode that should be used when “Program Full” is selected in the case of an audio input signal belonging to the digital (broadcast) signals category.

Return Values
Standard.

Precondition
```
aps.suppCategories[uhAprogSel_CategoryDigitalAudio]
```

Action
None.

Postcondition
```
*pOutputMode == aps.outputMode
```

7.2.6.5.2 uhIAprogSelNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in the audio program selector. The `Subscribe` function of the `UhIAprogSel` interface can be used to subscribe to these events.

Interface ID
```
uuid( A7D93188-05C3-400B-BAA2-65818FDA8C8A )
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.6.5.2.1 OnSubscriptionChanged

Signature

```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAprogSel_NtfSet_t notifs );
```

Qualifiers

— onsubscriptionchanged-function

7.2.6.5.2.2 OnInputSignalChanged

Signature

```
[async] Void OnInputSignalChanged (
    [in] UInt32 cookie,
    [in] uhAprogSel_InputSignal_t signal );
```

Qualifiers

None.

Description

This function is called by the program selector whenever it detects a change in the property of the audio input stream.

Parameters

Table 419

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`signal`	A structure providing the properties of the audio input stream.

Return Values

None.

Precondition

`True`

Action

— Any callback compliant action.

Postcondition

`True`

7.2.6.5.2.3 OnOutputSignalChanged

Signature

```
[async] Void OnOutputSignalChanged (
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32 cookie,
    [in] uhAprogSel_OutputSignal_t signal );
```

Qualifiers
None.

Description
This function is called by the program selector whenever a change occurs in the properties of the audio output stream.

Parameters

Table 420

Name	Description
cookie	The value that was passed by the client at subscription time.
signal	A structure providing properties of the audio output stream.

Return Values
None.

Precondition
True

Action
— Any callback compliant action.

Postcondition
True

7.2.6.5.2.4 OnActualProgramChanged

Signature
```
[async] Void OnActualProgramChanged (
    [in] UInt32 cookie,
    [in] uhAprogSel_Program_t program );
```

Qualifiers
None.

Description
This function is called by the program selector whenever the currently selected program for the audio input stream has changed.

Parameters

Table 421

Name	Description
cookie	The value that was passed by the client at subscription time.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
program	The new value of the selected program for the audio input stream.

Return Values
None.

Precondition
True

Action
— Any callback compliant action.

Postcondition
True

7.2.7 UhlAvolCtrl

7.2.7.1 Concepts

7.2.7.1.1 Introduction

This document defines the Audio Volume Control (AVC) logical component, and describes its user interfaces. Volume control is directly related to the sound level of the audio signal outputted by the audio playback system. This logical component can also provide the improvement functionality, termed Loudness-Processing, that ensures that the relative perceptual loudness levels for different frequencies are maintained across volume settings for the same musical piece. Mute control usually accompanies volume control, and interfaces for this functionality are also provided.

7.2.7.1.1.1 Loudness-Processing

Perceptual loudness of sound has a relationship to the ear's listening curves. In particular, when the volume is set to low levels, sounds at the very low and high frequencies often fall below the threshold of audibility. Loudness-Processing remedies this problem by boosting these frequency bands differently when the volume is set to low levels. As the volume setting is increased, the utility of Loudness-Processing steadily decreases. The volume setting is therefore a prime determinant in Loudness-Processing.

7.2.7.1.2 Operational Mode for AVC

The AVC LC comprises a filter with one logical input pin and one logical output pin.

The AVC provides a master volume setting, i.e a volume setting common to all audio channels in the input stream. Volume settings are allowed within a specific range only.

The on-off mute control, when provided, is at the stream level: the mute setting is the same for all the audio channels in the input stream.

Only an on-off control may be provided for Loudness-Processing through an optional interface.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.7.2 Types & Constants

7.2.7.2.1 Public Types & Constants

None.

7.2.7.2.2 Model Types & Constants

None.

7.2.7.3 Logical Component

7.2.7.3.1 Interface-Role Model

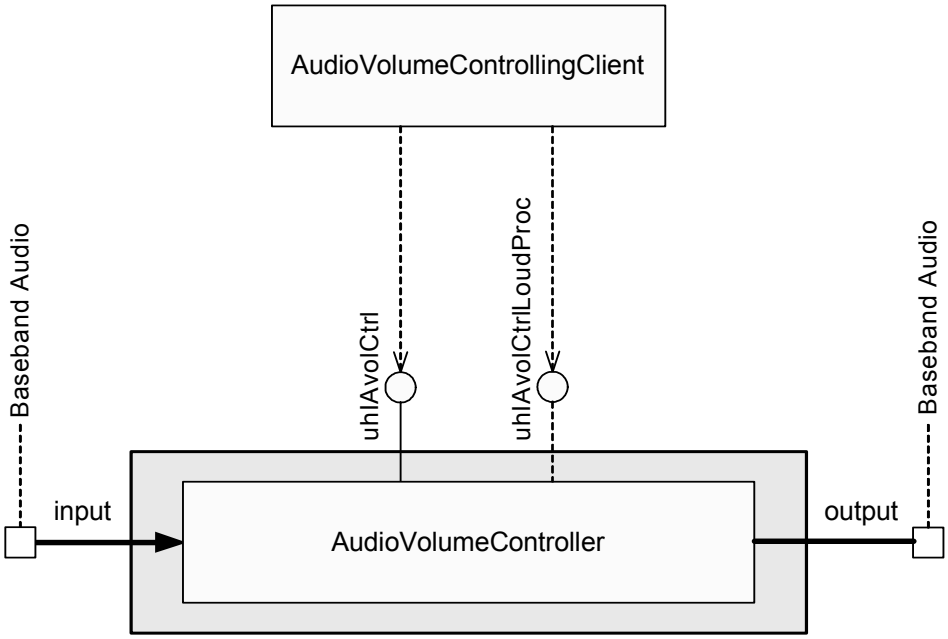


Figure 58 — Interface Role Model

The grey box in Figure 58 denotes an Audio Volume Control logical component containing a single `AudioVolumeController` role. This role provides one mandatory (solid line) control interface called `UhIAvolCtrl` and one optional interface called `uhIAvolCtrlLoudProc` to its client, the `AudioVolumeControllingClient`.

7.2.7.3.2 Diversity

7.2.7.3.2.1 Provided Interfaces

Table 422

Role	Interface	Presence
`AudioVolumeController`	`UhIAvolCtrl`	mandatory
	`uhIAvolCtrlLoudProc`	optional

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.7.3.2.2 Configurable Items

Table 423

Role	Attribute
AudioVolumeController	minVolSetting
	maxVolSetting
	muteSupported

7.2.7.3.2.3 Constraints

minVolSetting ≤ maxVolSetting

7.2.7.3.3 Instantiation

7.2.7.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 424

Type	Object	Multiplicity
AudioVolumeController	avc	1

7.2.7.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— avc.volume == minVolSetting
— if (muteSupported)
  — avc.mutedState == True
  else
  — avc.mutedState == False
  — avc.loudnessState == False

7.2.7.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.7.4 Roles

7.2.7.4.1 AudioVolumeController

Signature

```
role AudioVolumeController {
    const Int32    minVolSetting;
    const Int32    maxVolSetting;
    Int32          volume;
    const Bool     muteSupported;
    Bool           mutedState;
    Bool           loudnessState;
}
```

Qualifiers

— root

Description

The `AudioVolumeController` role performs the function of accepting and executing volume, mute and loudness settings from the user.

Independent Attributes

Table 425

Name	Description
`minVolSetting`	The minimum volume that can be set by the user. Minimum volume is not necessarily equivalent to mute activated; at minimum volume an audio signal can still be audible.
`maxVolSetting`	The maximum volume that can be set by the user.
`volume`	The master volume setting for the audio input stream, in integral steps. The user can make volume settings between (and including) `minVolSetting` and `maxVolSetting` in integral steps. How this is mapping into a physical volume parameter e.g. in dB, can be described in a platform instance document.
`muteSupported`	Describes whether mute control is supported by the `AudioVolumeController`.
`mutedState`	Describes whether the input stream is muted or not. It is `True` if the audio stream is muted; it is `False` if the audio stream is not muted.
`loudnessState`	Describes whether Loudness-processing is switched on or not. It is `True` if Loudness-processing is switched on; it is `False` if Loudness-processing is switched off.

Dependent Attributes

None.

Invariants

— `minVolSetting ≤ volume ≤ maxVolSetting`
— `If (!muteSupported)`
  — `mutedState == False`

Instantiation

An `AudioVolumeController` is always instantiated as part of an Audio Volume Control; see subclause 7.2.7.3.3.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Streaming Behavior
An `AudioVolumeController` has an audio signal as input stream. It adapts the signal level of the audio signal according to the volume set in `volume`. This volume setting applies for all the channels of the audio stream since the `AudioVolumeController` provides a master volume control. Volume settings are allowed within a specific range only, between (and including) `minVolSetting` and `maxVolSetting`.

When `loudnessState` is `True`, Loudness-Processing is also applied to the audio stream: sounds at the very low and high frequencies are boosted, the level of boosting being determined by the volume setting.

No output data is produced when the `AudioVolumeController` is in the state muted, i.e. when `mutedState` is `True`. When `mutedState` is `False`, the `AudioVolumeController` is not in the state muted and therefore audio signal is streamed out.

The `AudioVolumeController` takes care of a "smooth" transition while muting or un-muting, meaning that audible artefacts like "clicks" do not occur.

Active Behavior
None.

Execution Constraints
None.

7.2.7.4.2 AudioVolumeControllingClient

Signature
role `AudioVolumeControllingClient {}`

Qualifiers
— actor

Description
An Audio Volume Controlling Client represents a user of the `UhIAvolCtrl` and `uhIAvolCtrlLoudProc` interfaces.

Execution Constraints
None.

7.2.7.5 Interfaces

7.2.7.5.1 uhIAvolCtrl

Qualifiers
None.

Description
This interface of an AVC provides functions to set and retrieve volume and mute settings. Volume setting as well as muting apply to the entire stream, and not to individual channels.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( E82C5629-AD94-4BB1-AB0F-A2E0DD5E7C3C )`

7.2.7.5.1.1 GetVolRange

Signature
```
uhErrorCode_t GetVolRange (
    [out] pInt32 pMinVol,
    [out] pInt32 pMaxVol );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the range over which volume settings can be made.

Parameters

Table 426

Name	Description
`pMinVol`	Output parameter returning the minimum volume setting.
`pMaxVol`	Output parameter returning the maximum volume setting.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMinVol == avc.minVolSetting`
— `*pMaxVol == avc.maxVolSetting`

7.2.7.5.1.2 SetVol

Signature
```
uhErrorCode_t SetVol ( [in] Int32 volume );
```

Qualifiers
— synchronous

— single-theaded

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function sets the master volume setting. The volume setting can be made in integral steps in the range `[avc.minVolSetting, avc.maxVolSetting]`. Setting the volume will not unmute the audio, if it is currently muted.

Parameters

Table 427

Name	Description
`volume`	The master volume setting. How this is mapping into a physical volume parameter e.g. in dB, can be described in a platform instance document.

Return Values
Standard.

Precondition
`avc.minVolSetting ≤ volume ≤ avc.maxVolSetting`

Action
— Modify `avc.volume`

Postcondition
`avc.volume == volume`

7.2.7.5.1.3 GetVol

Signature
```
uhErrorCode_t GetVol ( [out] pInt32 pVolume );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the current master volume setting.

Parameters

Table 428

Name	Description
`pVolume`	Output parameter returning the current master volume setting. How this is mapping into a physical volume parameter e.g. in dB, can be described in a platform instance document.

Return Values
Standard.

Precondition
`True`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pVolume == avc.volume
```

7.2.7.5.1.4 GetMuteSupported

Signature
```
uhErrorCode_t GetMuteSupported ( [out] pBool pMuteSupported );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether mute control is supported by the AVC.

Parameters

Table 429

Name	Description
`pMuteSupported`	Output parameter returning whether mute control is supported by the AVC.

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
*pMuteSupported == avc.muteSupported
```

7.2.7.5.1.5 SetMute

Signature
```
uhErrorCode_t SetMute ( [in] Bool mute );
```

Qualifiers
— synchronous

— single-threaded

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Sets whether the audio input stream of the AVC should be muted or not.

Parameters

Table 430

Name	Description
`mute`	Defines whether the audio input stream of the AVC should be muted or not. It is `True` when the audio input stream should be muted. It is `False` when the audio input stream should not be muted.

Return Values
Standard.

Precondition
`avc.muteSupported == True`

Action
— Modify `avc.mutedState`

Postcondition
`avc.mutedState == mute`

7.2.7.5.1.6 GetMuted

Signature
```
uhErrorCode_t GetMuted ( [out] pBool pMuted );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the audio input stream of the AVC is currently muted or not.

Parameters

Table 431

Name	Description
`pMuted`	Output parameter that indicates whether the audio input stream of the AVC is currently muted or not. It is True when the audio input stream is muted. It is False when the audio input stream is not muted.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
`*pMuted == avc.mutedState`

7.2.7.5.2 uhlAvolCtrlLoudProc

Qualifiers
None.

Description
This interface provides control for switching-on (or off) Loudness-processing.

Interface ID
`uuid( BE7AAE44-F02A-4C04-BADC-6BB5DD070B45 )`

7.2.7.5.2.1 SetLoudProc

Signature
```
uhErrorCode_t SetLoudProc ( [in] Bool loudProc );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets whether Loudness-processing functionality should be switched on or off.

Parameters

Table 432

Name	Description
`loudProc`	Defines whether Loudness-processing functionality should be switched on or off. It is `True` when Loudness-processing functionality should be switched on. It is `False` when Loudness-processing functionality should be switched off.

Return Values
Standard.

Precondition
`True`

Action
— Modify `avc.loudnessState`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`avc.loudnessState == loudProc`

7.2.7.5.2.2 GetLoudProc

Signature
```
uhErrorCode_t GetLoudProc ( [out] pBool pLoudProc );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether Loudness-processing is switched on or off.

Parameters

Table 433

Name	Description
`pLoudProc`	Output parameter that indicates whether Loudness-processing is currently switched on or off. It is True when the Loudness-processing is currently switched on. It is False when Loudness-processing is currently switched off.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pLoudProc == avc.loudnessState`

7.2.8 UhlEqualizer

7.2.8.1 IConcepts

7.2.8.1.1 Introduction

Equalization allows the user to vary the tonal characteristics of the signal being reproduced in an audio chain or system. This is achieved by dividing the input signal into (overlapping) frequency bands and varying the gain in each of the frequency bands.

Depending on the platform realization, the channels of a multi-channel input might be grouped in so called channel sets. The channels within a set have the same frequency bands and the same adjustments (for these frequency bands). Different channel sets can have independent adjustments.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The interface specification defined in this document provides the ability to apply tonal control on a multi-channel audio input using an equalizer.

NOTE This interface does not provide an explicit Tone Control interface.However, Tone Control functionality can be realised using this interface via a 2 or 3 bands Equalizer.

7.2.8.2 Types & Constants

7.2.8.2.1 Public Types & Constants

7.2.8.2.1.1 Error Codes

None.

7.2.8.2.1.2 uhEqualizer_Channel_t

Signature

```
typedef enum _uhEqualizer_Channel_t {
    uhEqualizer_ChanLeft          = 0x00000001,
    uhEqualizer_ChanRight         = 0x00000002,
    uhEqualizer_ChanCenter        = 0x00000004,
    uhEqualizer_ChanSurround      = 0x00000008,
    uhEqualizer_ChanLeftSurround  = 0x00000010,
    uhEqualizer_ChanRightSurround = 0x00000020,
    uhEqualizer_ChanCenterSurround = 0x00000040,
    uhEqualizer_ChanLeftCenter    = 0x00000080,
    uhEqualizer_ChanRightCenter   = 0x00000100
} uhEqualizer_Channel_t,*puhEqualizer_Channel_t;
```

Qualifiers

— enum-element

Description

Values of this type represent channels of the audio input stream.

This type allows the user to identify a channel on which given set of gains can be applied for equalization.

Values

Table 434

Name	Description
uhEqualizer_ChanLeft	Left channel.
uhEqualizer_ChanRight	Right channel.
uhEqualizer_ChanCenter	Center channel.
uhEqualizer_ChanSurround	Surround channel.
uhEqualizer_ChanLeftSurround	Left surround channel.
uhEqualizer_ChanRightSurround	Right surround channel.
uhEqualizer_ChanCenterSurround	Center surround channel.
uhEqualizer_ChanLeftCenter	Left center channel.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhEqualizer_ChanRightCenter	Right center channel.

7.2.8.2.1.3 uhEqualizer_ChannelSet_t

Signature
```
typedef Int32 uhEqualizer_ChannelSet_t,*puhEqualizer_ChannelSet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of channels of the audio input stream.

This type allows the user to identify a set of channels on which specified gain can be applied for equalization.

Constraints
None.

7.2.8.2.1.4 uhEqualizer_Ntf_t

Signature
```
typedef enum _uhEqualizer_Ntf_t {
    uhEqualizer_SubscriptionChanged  = 0x00000001,
    uhEqualizer_GainChanged          = 0x00000002
} uhEqualizer_Ntf_t, *uhEqualizer_Ntf_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhIAdrcNtf.

Values

Table 435

Name	Description
`uhEqualizer_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhEqualizer_GainChanged`	Represents `OnGainChanged`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.8.2.1.5 uhEqualizer_NtfSet_t

Signature
```
typedef Int32 uhEqualizer_NtfSet_t, *puhEqualizer_NtfSet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type `uhEqualizer_Ntf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints
None.

7.2.8.2.1.6 uhEqualizer_AllNtfs

Signature
```
const uhEqualizer_NtfSet_t uhEqualizer_AllNtfs = 0x00000003;
```

Qualifiers
None.

Description
Defines the set of all values of type `uhEqualizer_Ntf_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

7.2.8.2.2 Model Types & Constants

None.

7.2.8.3 Logical Component

7.2.8.3.1 Interface-Role Model

Figure 59 depicts the interface-role model of the Equalizer Logical Component.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

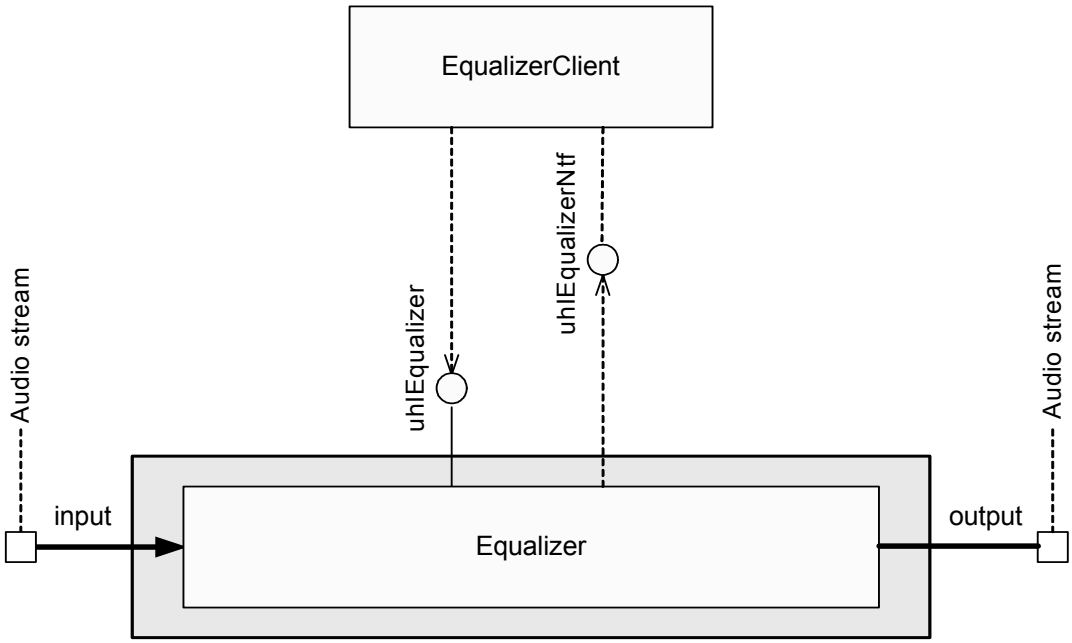


Figure 59 — Interface Role Model

The Equalizer Logical Component consists of the `Equalizer` role.

The role provides a control interface (`UhIEqualizer`).The `UhIEqualizer` interface allows the user to query the Equalizer parameters and to adjust the signal level of the audio input stream in defined frequency bands. This control interface is meant for use by the Equalizer client (represented by `EqualizerClient` role). An EqualizerClient can optionally provide call back interface `uhIEqualizerNtf`. This allows it to receive notifications of events in the Equalizer.

7.2.8.3.2 Diversity

7.2.8.3.2.1 Provided Interfaces

Table 436

Role	Interface	Presence
`Equalizer`	`UhIEqualizer`	mandatory
`EqualizerClient`	`uhIEqualizerNtf`	optional

7.2.8.3.2.2 Configurable Items

Table 437

Role	Attribute
`Equalizer`	`nrFreqBands`
	`minGain`
	`maxGain`
	`minFreq`
	`maxFreq`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Role	Attribute
	centerFreq
	maxFreqBandNum
	numSuppChannelSets
	pChannelSet

7.2.8.3.2.3 Constraints

```
—— numSuppChannelSets ≥ 1
—— maxFreqBandNum ≥ 1
—— minGain ≤ maxGain
—— For each c1 in {0 .. numSuppChannelSets −1}
   —— For each c2 in {0 .. numSuppChannelSets −1}
      —— If c1 != c2
         —— pChannelSet[c1] & pChannelSet[c2] = 0
—— For each c in {0 .. numSuppChannelSets −1}
   —— 1 ≤ nrFreqBands[c] ≤ maxFreqBandNum
—— For each c in {0 .. numSuppChannelSets −1}
   —— For each i in {0 .. nrFreqBands[c] −1}
      —— minFreq[c][i]    ≤ centerFreq[c][i]
      —— centerFreq[c][i] ≤ maxFreq[c][i]
      —— minFreq[c][i]    ≤ minFreq[c][i+1]
      —— centerFreq[c][i] ≤ centerFreq[c][i+1]
      —— maxFreq[c][i]    ≤ maxFreq[c][i+1]
```

7.2.8.3.3 Instantiation

7.2.8.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 438

Type	Object	Multiplicity
Equalizer	eq	1

7.2.8.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
—— For each c in {0 .. numSuppChannelSets −1}
   —— For each i in {0 .. nrFreqBands[c] −1}
      —— gain[c][i] == minGain
```

7.2.8.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.8.4 Roles

7.2.8.4.1 Equalizer

Signature

```
role Equalizer {
    const UInt32    numSuppChannelSets;
    const uhEqualizer_ChannelSet_t    pChannelSet[numSuppChannelSets];
    const UInt32    nrFreqBands[numSuppChannelSets];
    const UInt32    maxFreqBandNum;
    const UInt32    minFreq[numSuppChannelSets][maxFreqBandNum];
    const UInt32    centerFreq[numSuppChannelSets][maxFreqBandNum];
    const UInt32    maxFreq[numSuppChannelSets][maxFreqBandNum];
    const Int32     minGain;
    const Int32     maxGain;
    Int32           gain[numSuppChannelSets][maxFreqBandNum];
}
```

Qualifiers

— root

Description

The `Equalizer` role adjusts the signal level of the audio input stream in defined frequency bands. Depending on the platform realization, the channels of a multi-channel input might be grouped in so called channel sets. The channels within a set have the same frequency bands and the same adjustments (for these frequency bands). Different channel sets can have independent adjustments.

Independent Attributes

Table 439

Name	Description
`numSuppChannelSets`	Number of supported channel-sets.
`pChannelSet`	Array of length `numSuppChannelSets` containing the channel-sets provided by the platform instance.
`nrFreqBands`	Array containing the number of frequency bands for every channel-set.
`maxFreqBandNum`	Maximum number of supported frequency bands that can possibly occur in the equalizer for any channel-set.
`minFreq`	Lower frequency cutoff per frequency band and per channel-set.
`centerFreq`	Centre frequency value per frequency band and per channel-set.
`maxFreq`	Higher frequency cutoff per frequency band and per channel-set.
`minGain`	The minimum gain that can be applied to the audio input stream of the Equalizer. How the integer gain value translates into physical gain value for example in dB, will be described in the platform instance document.
`maxGain`	The maximum gain that can be applied to the audio input stream of the Equalizer. How the integer gain value translates into physical gain value for example in dB, will be described in the platform instance document.
`gain`	Current gains associated with a frequency band and channel-set. How the integer gain value translates into physical gain value for example in dB, will be described in the platform instance document.

Dependent Attributes

None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Invariants

```
— For each c in {0 .. numSuppChannelSets −1}
   — For each i in {0 to nrFreqBands[c] −1}
      — minGain ≤ gain[c][i] ≤ maxGain
```

Instantiation
See subclause 7.2.8.3.3

Streaming Behavior
`Equalizer` has an audio signal as input stream. Signal frequency components inside equalizer frequency band, for the specified channels are adjusted (attenuated or amplified) based on the `Equalizer` gain settings and audio data is sent on the output streaming path.

Active Behavior
The main event associated with the `Equalizer` role is as follows:

Table 440

Event	Action
Whenever `Equalizer` detects a change in the gain for a channel-set and a frequency band, it notifies the `EqualizerClient` (see `uhIEqualizerNtf`). Such a change is necessarily the effect of an equalizer setting made by the user; it does not autonomously originate from the stream.	`OnGainChanged`

Execution Constraints
None.

7.2.8.4.2 EqualizerClient

Signature

```
role EqualizerClient {}
```

Qualifiers
— actor

Description
An Equalizer Client represents a user of `UhIEqualizer` interface.

Execution Constraints
None.

7.2.8.5 Interfaces

7.2.8.5.1 uhIEqualizer

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface provides functions to query the `Equalizer` parameters and control the gain applied on the input stream.

Interface ID
`uuid(37B9694E-15F9-47F6-92A3-CB0D91ADE964)`

7.2.8.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIEqualizerNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhEqualizer_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

7.2.8.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe(
    [in] uhIEqualizerNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhEqualizer_NtfSet_t notifs );
```

Qualifiers
— unsubscribe-function.

7.2.8.5.1.3 GetNumSuppChannelSets

Signature
```
uhErrorCode_t GetNumSuppChannelSets ([out] pUInt32 pNumSuppChannelSets);
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the number of supported channel-sets

Parameters

Table 441

Name	Description
`pNumSuppChannelSets`	Output parameter returning the number of supported channel-sets.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
`*pNumSuppChannelSets == eq.numSuppChannelSets`

7.2.8.5.1.4 GetSuppChannelSets

Signature
```
uhErrorCode_t GetSuppChannelSets (
    [in] UInt32 number,
    [out, size_is(number)] puhEqualizer_ChannelSet_t pChannelSet );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the supported channel-sets.

Parameters

Table 442

Name	Description
`number`	Parameter specifying the number of elements in the output channel-set array.
`pChannelSet`	Output parameter returning a pointer to an array with supported channel-sets.

Return Values
Standard.

Precondition
`number ≥ eq.numSuppChannelSets`

Action
None.

Postcondition
— For each c in {0 .. eq.numSuppChannelSets −1}
  — *pChannelSet[c] == eq.pChannelSet[c]
— For each c in {eq.numSuppChannelSets .. number −1}
  — *pChannelSet[c] == 0

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.8.5.1.5 GetNrFreqBands

Signature

```
uhErrorCode_t GetNrFreqBands(
    [in] uhEqualizer_ChannelSet_t  channelSet,
    [out] pUInt32                  pNrFreqBands);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the number of frequency bands available for the specified channel set in the Equalizer.

Parameters

Table 443

Name	Description
`channelSet`	Channel-set for which the number of frequency bands is requested.
`pNrFreqBands`	Output parameter returning the number of frequency bands.

Return Values

Standard.

Precondition

`channelSet in eq.pChannelSet[]`

Action

None.

Postcondition

— Let `c` be the index of `channelSet` in `eq.pChannelSet[]`

— `*pNrFreqBands == eq.nrFreqBands[c]`

7.2.8.5.1.6 GetBandRange

Signature

```
uhErrorCode_t GetBandRange(
    [in] uhEqualizer_ChannelSet_t  channelSet,
    [in] UInt32                    freqBand,
    [out] pUInt32                  pMinFreq,
    [out] pUInt32                  pMaxFreq,
    [out] pUInt32                  pCenterFreq);
```

Qualifiers

— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the minimum and maximum range as well as the center frequency for the specified frequency band and channel-set.

Parameters

Table 444

Name	Description
`channelSet`	Channel-set for which the frequency range is requested.
`freqBand`	Frequency band for which the frequency range is requested.
`pMinFreq`	Output parameter specifying the minimum frequency value for the specified frequency band.
`pMaxFreq`	Output parameter specifying the maximum frequency value for the specified frequency band.
`pCenterFreq`	Output parameter specifying the center frequency value for the specified frequency band.

Return Values
Standard.

Precondition
— `channelSet` in `eq.pChannelSet[]`
— Let `c` be the index of `channelSet` in `eq.pChannelSet[]`
— `0 ≤ freqBand < eq.nrFreqBands[c]`

Action
None.

Postcondition
— `*pMinFreq == eq.minFreq[c][freqBand]`
— `*pMaxFreq == eq.maxFreq[c][freqBand]`
— `*pCenterFreq == eq.centerFreq[c][freqBand]`

7.2.8.5.1.7 GetGainRange

Signature
```
uhErrorCode_t GetGainRange(
    [out] pInt32  pMinGain,
    [out] pInt32  pMaxGain);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum value of the gain of the Equalizer.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 445

Name	Description
pMinGain	Output parameter that indicates the minimum gain that can be applied to the audio input stream of the Equalizer. How the integer gain value translates into physical gain value for example in dB, will be described in the platform instance document.
pMaxGain	Output parameter that indicates the maximum gain that can be applied to the audio input stream of the Equalizer. How the integer gain value translates into physical gain value for example in dB, will be described in the platform instance document.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pMinGain == eq.minGain
— *pMaxGain == eq.maxGain

7.2.8.5.1.8 SetGain

Signature

```
uhErrorCode_t SetGain (
    [in] uhEqualizer_ChannelSet_t  channelSet,
    [in] UInt32                    freqBand,
    [in] Int32                     gain );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the gain for the specified channel-set in the specified frequency band.

Parameters

Table 446

Name	Description
channelSet	The channel-set for which to set the gain.
freqBand	The frequency band for which to set the gain.
gain	The gain value to be applied to the channels of the specified channel-set and the specified frequency band. How the integer gain value translates into physical gain value for example in dB, will be described in the platform instance document.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
— `channelSet in eq.pChannelSet[]`
— `Let c be the index of channelSet in eq.pChannelSet[]`
— `0 ≤ freqBand < eq.nrFreqBands[c]`
— `eq.minGain ≤ gain ≤ eq.maxGain`

Action
`Modify eq.gain[c][freqBand]`

Postcondition
`eq.gain[c][freqBand] == gain`

7.2.8.5.1.9 GetGain

Signature
```
uhErrorCode_t GetGain(
    [in]  uhEqualizer_ChannelSet_t  channelSet,
    [in]  UInt32                    freqBand,
    [out] pInt32                    pGain);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the gain for the specified channel-set and the specified frequency band.

Parameters

Table 447

Name	Description
`channelSet`	The channel-set for which the gain is requested.
`freqBand`	The frequency band for which the gain is requested.
`pGain`	Output parameter returning the gain value applied to the channels of the specified channel-set and the specified frequency band. How the integer gain value translates into physical gain value for example in dB, will be described in the platform instance document.

Return Values
Standard.

Precondition
— `channelSet in eq.pChannelSet[]`
— Let `c` be the index of `channelSet` in `eq.pChannelSet[]`
— `0 ≤ freqBand < eq.nrFreqBands[c]`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
`*pGain == eq.gain[c][freqBand]`

7.2.8.5.2 uhIEqualizerNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in an `Equalizer`. The Subscribe function of `UhIEqualizer` can be used to subscribe to these events.

Interface ID
`uuid(9FE97C31-2AEA-46A3-9113-5B811CE3E6EC)`

7.2.8.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged(
    [in] UInt32 cookie,
    [in] uhEqualizer_NtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

7.2.8.5.2.2 OnGainChanged

Signature
```
[async] Void OnGainChanged (
    [in] UInt32                    cookie,
    [in] uhEqualizer_ChannelSet_t  channelSet,
    [in] UInt32                    freqBand,
    [in] Int32                     gain);
```

Qualifiers
None.

Description
This function is called by the `Equalizer` to notify a client that the gain of a frequency band of a channel-set has changed.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 448

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`channelSet`	The channel-set for which the gain has changed.
`freqBand`	The frequency band for which the gain has changed.
`gain`	The new gain value.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

7.2.9 UhlOutConf

See 6.1.4.

7.2.10 UhlSpkr

7.2.10.1 Concepts

7.2.10.1.1 Introduction

The current specification details the software interfaces for the Speaker-set logical component. The following subclauses detail the basics of the construct.

7.2.10.1.1.1 Speaker Pins

A speaker set is a set of loudspeakers that have a functional relation, for example: the 5.1-channel speaker installation, or the headphones. In the current context, a Speaker-set is the logical component (LC) that acts as a ***sink in the audio playback chain***. Since the actual speakers are located external to the audio processing platform, each speaker is represented as a "speaker pin" (SP) internal to the Speaker-set LC. A speaker pin therefore has attributes that are settable by the user.

7.2.10.1.1.2 Functionality Associated with the Speaker-set

The Speaker-set LC provides the following functionality and associated control for inputs into speaker pins:

— **Bass-redirection**: direction of low frequency audio into speaker pins that support low-frequency playback [6]. Bass redirection has an on-off control for operation. The LC implementation will use internal logic to choose an appropriate bass-redirection scheme for the given speaker capabilities and the input stream configuration. This internal logic may be compliant with bass redirection modes in use in the industry (e.g.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the Dolby Digital bass-redirection modes) or not. The LC therefore provides the user with interfaces to specify speaker capabilities1[11], but explicit bass-redirection modes e.g. corresponding to the Dolby specifications are not programmable by the user. Further details are provided in subclause Bass Redirection.

— **Trim control**: a boost or attenuation provided per speaker pin to fine-tune speaker output levels during set-up, and to provide differential volume setting (between speakers) at run-time.

— **Delay**: To compensate for speaker placement in appropriate use context. The delay setting is provided to the user under the assumption that the user has the relevant knowledge for his choice of parameters.

The model for this logical component is illustrated in Figure 60. The association of input channels to speaker pins is a fixed one, see Table 449, and becomes active at the input of the bass-redirection module although the physical connection to the speaker pins happens further downstream. What this essentially means is that bass-redirection, trim and delay settings are speaker pin specific rather than being channel-specific for all practical purposes.

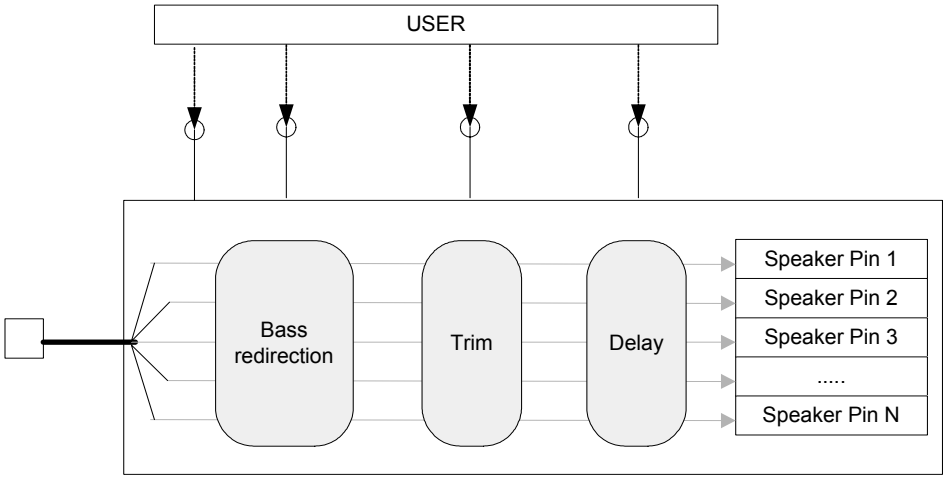


Figure 60 — Functional model for the Speaker-set / headphones LC. Trim and delay are logically executed after bass redirection. The horizontal lines denote audio channels. The interface notations are to be taken notionally to understand control links.

Table 449 — Channel-to-speaker association

Channel	Speaker Pin	Speaker Pin Description
Left	`uhSpkr_SpkrLeft`	Left speaker pin.
Right	`uhSpkr_SpkrRight`	Right speaker pin.
Center	`uhSpkr_SpkrCenter`	Center speaker pin.
Surround	`uhSpkr_SpkrSurround`	Surround speaker pin.
Left surround	`uhSpkr_SpkrLeftSurround`	Left surround speaker pin.
Right surround	`uhSpkr_SpkrRightSurround`	Right surround speaker pin.
Center surround	`uhSpkr_SpkrCenterSurround`	Center surround speaker pin.

11 This is an implicit way of specifying bass redirection requirements.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Channel	Speaker Pin	Speaker Pin Description
Left center	uhSpkr_SpkrLeftCenter	Left center speaker pin.
Right center	uhSpkr_SpkrRightCenter	Right center speaker pin.
Lfe	uhSpkr_SpkrSubWoofer	Sub-woofer speaker pin.

Trim Control and Delay

Trim control and delay are set for a speaker system for a given room placement during set-up. This allows the user to get best surround effects over a central region commonly termed as the "sweet spot", where listeners are expected to be seated. During the set-up of the speakers, the trim (and delay) settings can be performed: this requires the bass-redirection processing to be turned off. There is no autonomous deactivation (and reactivation) of the bass-redirection processing. It is up to the client to deactivate the bass-redirection processing (if at that moment it is applied) before trim settings are done and to reactive it (if it was previously applied) after trim settings are done.

Trim control doubles up as a mechanism to provide differential volume control between speakers over and above a master volume control. This is because trim settings are made for specific speaker pins, and it is therefore possible to derive "balance" controls or separate volume settings for speaker-pairs from these at the level of the end-user.

Bass Redirection

Bass redirection involves direction of the bass signal component from audio channels to speakers that can play it. The speaker configuration is sufficient for determining how bass redirection should be done. E.g., if a 5.1 channel stream is played though a speaker configuration comprising 5 satellite speakers (mid-range) and a sub-woofer, bass redirection would involve collecting the low-frequency signals from all the 5 channels, adding these to the LFE channel and (re)directing the result into the sub-woofer. For the current LC, speaker characterization is therefore clubbed into the bass redirection interface.

Characterization of generic bass-redirection modes (i.e. the routing of bass to speakers that can play the same) may be made independently by a platform provider in the platform instance document or may be compliant with modes in use in the industry. The most common example of bass redirection modes in use in the industry are the ones from Dolby, detailed in Appendix D of the Dolby licensee information manual [22][12].

7.2.10.2 Types & Constants

7.2.10.2.1 Public Types & Constants

7.2.10.2.1.1 Error Codes

None.

12 The Dolby LIM uses the term "large" to denote speakers that can play the full range of audio signals, and "small" to denote speakers that cannot play frequencies below 120 Hz. The corresponding terminology in this specification comprises "full-range" and "mid-range" respectively.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.10.2.1.2 uhSpkr_SpkrPinIndex_t

Signature

```
typedef enum _uhSpkr_SpkrPinIndex_t {
    uhSpkr_SpkrLeft           = 0x00000001,
    uhSpkr_SpkrRight          = 0x00000002,
    uhSpkr_SpkrCenter         = 0x00000004,
    uhSpkr_SpkrSurround       = 0x00000008,
    uhSpkr_SpkrLeftSurround   = 0x00000010,
    uhSpkr_SpkrRightSurround  = 0x00000020,
    uhSpkr_SpkrCenterSurround = 0x00000040,
    uhSpkr_SpkrLeftCenter     = 0x00000080,
    uhSpkr_SpkrRightCenter    = 0x00000100,
    uhSpkr_SpkrSubWoofer      = 0x00000200
} uhSpkr_SpkrPinIndex_t, *puhSpkr_SpkrPinIndex_t;
```

Qualifiers

— enum-element

Description

This type is used to denote the speaker pins. It is assumed that a maximum of 10 speaker pins needs to be addressed.

Values

Table 450

Name	Description
uhSpkr_SpkrLeft	Left speaker pin.
uhSpkr_SpkrRight	Right speaker pin.
uhSpkr_SpkrCenter	Center speaker pin.
uhSpkr_SpkrSurround	Surround speaker pin.
uhSpkr_SpkrLeftSurround	Left surround speaker pin.
uhSpkr_SpkrRightSurround	Right surround speaker pin.
uhSpkr_SpkrCenterSurround	Center surround speaker pin.
uhSpkr_SpkrLeftCenter	Left center speaker pin.
uhSpkr_SpkrRightCenter	Right center speaker pin.
uhSpkr_SpkrSubWoofer	Sub-woofer speaker pin.

7.2.10.2.1.3 uhSpkr_SpkrPinIndexSet_t

Signature

```
typedef Int32
uhSpkr_SpkrPinIndexSet_t, *p
uhSpkr_SpkrPinIndexSet_t;
```

Qualifiers

— enum-set

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of speaker pins.

Constraints
None.

7.2.10.2.1.4 uhSpkr_SpkrType_t

Signature
```
typedef enum _uhSpkr_SpkrType_t {
    uhSpkr_TypeFullRange = 0,
    uhSpkr_TypeMidRange  = 1,
    uhSpkr_TypeSubWoofer = 2
} uhSpkr_SpkrType_t, *puhSpkr_SpkrType_t;
```

Qualifiers
None.

Description
This type is used to denote the speaker frequency response range.

Values

Table 451

Name	Description
uhSpkr_TypeFullRange	Denotes speakers that can handle the entire range of audio signals (20-20000 Hz), including the sub-woofer range (less than 120 Hz). These are referred to as "large" in Dolby context, because of their usual size.
uhSpkr_TypeMidRange	Denotes speakers that have impaired capabilities in the sub-woofer range of frequencies (less than 120 Hz). These are referred to as "small" in Dolby context.
uhSpkr_TypeSubWoofer	Sub-woofer, catering to the very low frequency range only. Usually associated with the LFE component in multi-channel audio.

Remarks
— In terms of bass-redirection, the sub-woofer can only play 120 Hz low-pass filtered audio signals, the mid-range speaker 120 Hz high-pass filtered signals and the full-range speaker can play the complete signal.

7.2.10.2.1.5 uhSpkr_BassRedirStdCompliancy_t

Signature
```
typedef enum _uhSpkr_BassRedirStdCompliancy_t {
    uhSpkr_StdNone  = 0,
    uhSpkr_StdDolby = 1
} uhSpkr_BassRedirStdCompliancy_t, *puhSpkr_BassRedirStdCompliancy_t;
```

Qualifiers
None.

Description
Values of this type represent standards to which the mapping between the speaker configuration and the routing of the bass signal is compliant.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 452

Name	Description
uhSpkr_StdNone	The mapping between the speaker configuration and the routing of the bass signal does not follow any specific standard. The actual mapping is determined by the platform implementation and may be indicated in the related platform instance document.
uhSpkr_StdDolby	The mapping between the speaker configuration and the routing of the bass signal follows the specification of the Dolby Digital standard; see Appendix D of [22] for details over the bass redirection mode corresponding to a specific speaker configuration.

7.2.10.2.2 Model Types & Constants

7.2.10.2.2.1 SpkrAttrib

Signature

```
typedef struct {
    uhSpkr_SpkrType_t  speakerType;
    Int32              trim;
    UInt32             delay;
    Bool               active;
} SpkrAttrib;
```

Qualifiers
None.

Description
This structure captures the attributes associated with each speaker pin.

Fields

Table 453

Name	Description
speakerType	The type of speaker connected to the speaker pin in terms of frequency response.
trim	The trim setting of the speaker pin in integral steps.
delay	The delay setting associated with the speaker pin in integral steps.
active	This value indicates whether the speaker pin is active (i.e. in use) in the LC instance. When it is `True` the speaker pin is active (i.e. in use) and audio data can be outputted on it. When it is `False` the speaker pin is not active (i.e. not in use) and no audio data can be outputted on it.

Constraints
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.10.3 Logical Component

7.2.10.3.1 Interface-Role Model

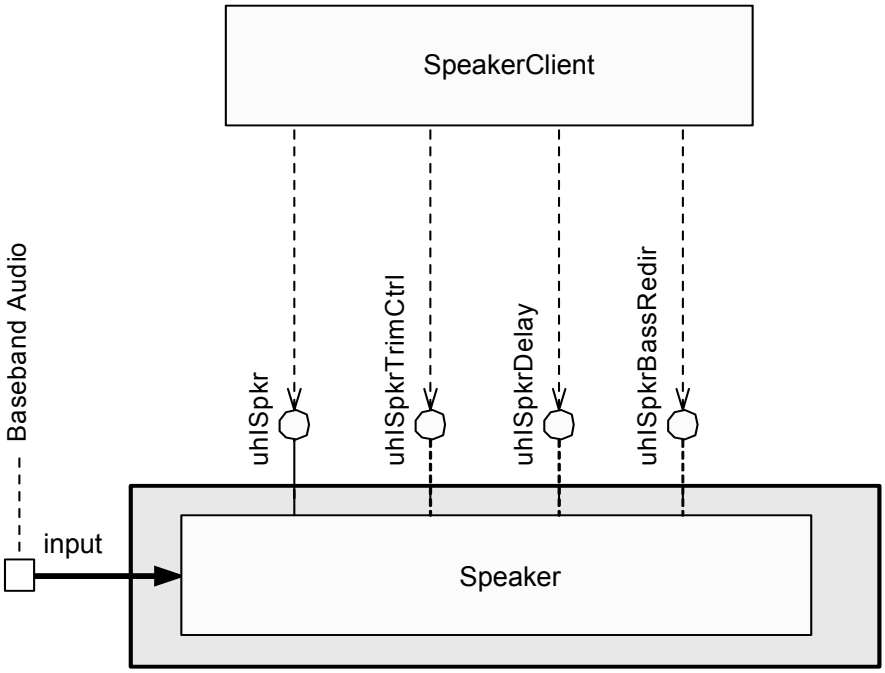


Figure 61 — Interface Role Model for Speaker-set logical component.

The grey box in Figure 61 denotes a Speaker-set logical component containing a single `Speaker` role.

This role provides one mandatory (solid line) control interface called `UhISpkr` and three optional interfaces called `uhISpkrTrimCtrl`, `uhISpkrDelay` and

`uhISpkrBassRedir` to its client, the `SpeakerClient`.

7.2.10.3.2 Diversity

7.2.10.3.2.1 Provided Interfaces

Table 454

Role	Interface	Presence
`Speaker`	`UhISpkr`	mandatory
	`uhISpkrTrimCtrl`	optional
	`uhISpkrDelay`	optional
	`uhISpkrBassRedir`	optional

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.10.3.2.2 Configurable Items

Table 455

Role	Attribute
Speaker	suppSpeakerPins
	minTrim
	maxTrim
	minDelay
	maxDelay
	bassRedirStdCompliancy

7.2.10.3.2.3 Constraints

- minTrim ≤ maxTrim
- minDelay ≤ maxDelay
- suppSpeakerPins != 0x00000000

7.2.10.3.3 Instantiation

7.2.10.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 456

Type	Object	Multiplicity
Speaker	spkr	1

7.2.10.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

- For each j in uhSpkr_SpkrPinIndex_t
  - spkr.pSpeakers[j].speakerType == uhSpkr_TypeFullRange
  - spkr.pSpeakers[j].trim == minTrim
  - spkr.pSpeakers[j].delay == minDelay
  - spkr.pSpeakers[j].active == False
  - spkr.currentBassRedirectionState == False

7.2.10.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.10.4 Roles

7.2.10.4.1 Speaker

Signature

```
role Speaker {
    const
uhSpkr_SpkrPinIndexSet_t          suppSpeakerPins;
    SpkrAttrib                              pSpeakers[uhSpkr_SpkrPinIndex_t];
    const Int32                             minTrim;
    const Int32                             maxTrim;
    const UInt32                            minDelay;
    const UInt32                            maxDelay;
    Bool                                    currentBassRedirectionState;
    const uhSpkr_BassRedirStdCompliancy_t   bassRedirStdCompliancy;
}
```

Qualifiers

— root

Description

The `Speaker` role provides (de)activation of individual speaker pins. In addition, it provides interfaces for bass-redirection, trim control and delay functionality[13].

Independent Attributes

Table 457

Name	Description
`suppSpeakerPins`	The set of speaker pins that are available and for which activation is supported.
`pSpeakers`	Array containing speaker pin attributes. Each element of this array corresponds to one speaker pin.
`minTrim`	The minimum trim setting possible for the speaker pins. The actual translation of this value into a gain (e.g. in dB) will be provided in the platform instance document.
`maxTrim`	The maximum trim setting possible for the speaker pins. The actual translation of this value into a gain (e.g. in dB) will be provided in the platform instance document.
`minDelay`	The minimum delay setting possible for the speaker pins. The actual translation of this value into an absolute delay (e.g. in ms) will be provided in the platform instance document.
`maxDelay`	The maximum delay setting possible for the speaker pins. The actual translation of this value into an absolute delay (e.g. in ms) will be provided in the platform instance document.
`currentBassRedirectionState`	The current bass redirection state i.e. whether bass redirection is active. It is `True` when bass redirection is active. It is `False` when bass redirection is not active.
`bassRedirStdCompliancy`	The standard to which the mapping between the speaker configuration and the routing of the bass signal is compliant.

13 Details of these functions are provided in subclause 7.2.10.1.1.2.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes
None.

Invariants
— For each j in uhSpkr_SpkrPinIndex_t
  — If j in suppSpeakerPins
    — minTrim ≤ pSpeakers[j].trim ≤ maxTrim
    — minDelay ≤ pSpeakers[j].delay ≤ maxDelay
  — else
    — pSpeakers[j].speakerType == uhSpkr_TypeFullRange
    — pSpeakers[j].trim == minTrim
    — pSpeakers[j].delay == minDelay
    — pSpeakers[j].active == False

Instantiation
A `Speaker` role is always instantiated as part of a Speaker-set logical component; see subclause 7.2.10.3.3.

Streaming Behavior
The `Speaker` role has only one streaming input and no streaming output. The input stream contains uncompressed audio data.

The `Speaker` role applies if necessary bass redirection, trim and/or delay to the audio input stream.

The `Speaker` role has one or more speaker pins. Audio channels of the audio input stream are coupled to specific speaker pins according to a fixed scheme, see Table 449.

When a speaker pin is inactive, no audio data can be outputted on it.

When a speaker pin is active, audio data is only outputted when the associated channel contains data, otherwise the speaker pin is muted and no data is outputted.

Speakers can be connected to the speaker pins to reproduce the processed audio stream.

Active Behavior
None.

Execution Constraints
None.

7.2.10.4.2 SpeakerClient

Signature
role SpeakerClient {}

Qualifiers
— actor

Description
A Speaker Client represents a user of the `UhISpkr` (and optionally of `uhISpkrTrimCtrl`, `uhISpkrDelay` and

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

uhISpkrBassRedir) interface.

Execution Constraints
None.

7.2.10.5 Interfaces

7.2.10.5.1 uhISpkr

Qualifiers
None.

Description
This interface is used to (de)activate speaker pins.

Interface ID
uuid(E7E1DADC-1E42-47B3-B808-B35849AFBA02)
Remarks

— The logical model follows the depiction in Figure 60.

7.2.10.5.1.1 GetSuppSpkrPins

Signature
uhErrorCode_t GetSuppSpkrPins ([out] p
uhSpkr_SpkrPinIndexSet_t pSuppSpkrPins);

Qualifiers
— synchronous

— thread-safe

Description
This function returns the set of speaker pins that are available and for which activation is supported.

Parameters

Table 458

Name	Description
pSuppSpkrPins	Output parameter returning the set of speaker pins that are available and for which activation is supported.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
`*pSuppSpkrPins == spkr.suppSpeakerPins`

7.2.10.5.1.2 SetSpkrActive

Signature
```
uhErrorCode_t SetSpkrActive (
    [in] uhSpkr_SpkrPinIndex_t spkrPinIndex,
    [in] Bool active );
```

Qualifiers
— synchronous

— single-threaded

Description
This function activates or deactivates a given speaker pin in the LC instance.

Parameters

Table 459

Name	Description
`spkrPinIndex`	The speaker pin for which the activation/deactivation is queried.
`active`	Variable specifying the speaker pin activation.

Return Values
Standard.

Precondition
`(spkr.suppSpeakerPins & spkrPinIndex) == spkrPinIndex`

Action
— Modify `spkr.pSpeakers[spkrPinIndex].active`

Postcondition
`spkr.pSpeakers[spkrPinIndex].active == active`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.10.5.1.3 GetSpkrActive

Signature

```
uhErrorCode_t GetSpkrActive (
    [in] uhSpkr_SpkrPinIndex_t spkrPinIndex,
    [out] pBool pActive );
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns whether a given speaker pin is active in the LC instance.

Parameters

Table 460

Name	Description
`spkrPinIndex`	The speaker pin for which the current activation/deactivation state is queried.
`pActive`	Output parameter stating whether the speaker pin is activated.

Return Values

Standard.

Precondition

```
(spkr.suppSpeakerPins & spkrPinIndex) == spkrPinIndex
```

Action

None.

Postcondition

```
*pActive == spkr.pSpeakers[spkrPinIndex].active
```

7.2.10.5.2 uhlSpkrTrimCtrl

Qualifiers

None.

Description

This interface is used to make and query trim settings for the speaker set[14]. Trim settings are individualized to speaker pins.

14 Refer to subclause 7.2.10.1.1.2 Trim Control and Delay.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 956D698A-13D2-44C2-8BA0-78E19617B243 )`

7.2.10.5.2.1 GetTrimRange

Signature

```
uhErrorCode_t GetTrimRange (
    [out] pInt32 pMinTrim,
    [out] pInt32 pMaxTrim );
```

Qualifiers

— synchronous

— thread-safe

Description
This function returns the range over which trim settings can be made in integral steps. The actual trim values (e.g. in dB) for this range will be specified in the platform instance document.

Parameters

Table 461

Name	Description
`pMinTrim`	Output parameter specifying the minimum trim setting.
`pMaxTrim`	Output parameter specifying the maximum trim setting.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition

— `*pMinTrim == spkr.minTrim`
— `*pMaxTrim == spkr.maxTrim`

Remarks

— The range is common to all speaker pins.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.10.5.2.2 SetTrim

Signature

```
uhErrorCode_t SetTrim (
    [in] uhSpkr_SpkrPinIndex_t spkrPinIndex,
    [in] Int32 trim );
```

Qualifiers

— synchronous

— single-threaded

Description

This function sets the trim for a given speaker pin in integral steps in the range `[spkr.minTrim,spkr.maxTrim]`.

Parameters

Table 462

Name	Description
`spkrPinIndex`	The speaker pin for which trim is set.
`trim`	The trim setting specified in integral steps in the range `[spkr.minTrim,spkr.maxTrim]`.

Return Values

Standard.

Precondition

— `(spkr.suppSpeakerPins & spkrPinIndex) == spkrPinIndex`

— `spkr.minTrim ≤ trim ≤ spkr.maxTrim`

Action

— Modify `spkr.pSpeakers[spkrPinIndex].trim`

Postcondition

— `spkr.pSpeakers[spkrPinIndex].trim == trim`

7.2.10.5.2.3 GetTrim

Signature

```
uhErrorCode_t GetTrim (
    [in] uhSpkr_SpkrPinIndex_t spkrPinIndex,
    [out] pInt32 pTrim );
```

Qualifiers

— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the current trim setting for a given speaker pin.

Parameters

Table 463

Name	Description
`spkrPinIndex`	The speaker pin for which the trim setting is queried.
`pTrim`	Output parameter specifying the current trim setting in integral steps in the range `[spkr.minTrim,spkr.maxTrim]`.

Return Values
Standard.

Precondition
```
(spkr.suppSpeakerPins & spkrPinIndex) == spkrPinIndex
```

Action
None.

Postcondition
```
*pTrim == spkr.pSpeakers[spkrPinIndex].trim
```

7.2.10.5.3 uhlSpkrDelay

Qualifiers
None.

Description
This interface is used to make and query delay settings for the speaker set. Delay settings are individualized to speaker pins based on speaker placement.

Interface ID
```
uuid( BEDE30FA-BD66-4176-9009-F4A6FEE1AAA2 )
```

7.2.10.5.3.1 GetDelayRange

Signature
```
uhErrorCode_t GetDelayRange (
    [out] pUInt32 pMinDelay,
    [out] pUInt32 pMaxDelay );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the range of delay settings that can be made on the path to a speaker pin in integral steps. The actual delay values (e.g. in ms) for this range will be specified in the platform instance document.

Parameters

Table 464

Name	Description
`pMinDelay`	Output parameter specifying the minimum delay setting.
`pMaxDelay`	Output parameter specifying the maximum delay setting.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMinDelay == spkr.minDelay`
— `*pMaxDelay == spkr.maxDelay`

Remarks

— The range is common to all speaker pins.

7.2.10.5.3.2 SetDelay

Signature
```
uhErrorCode_t SetDelay (
    [in] uhSpkr_SpkrPinIndex_t spkrPinIndex,
    [in] UInt32 delay );
```

Qualifiers
— synchronous
— single-threaded

Description
This function sets a delay on the path to a given speaker pin in integral steps in the range `[spkr.minDelay,spkr.maxDelay]`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 465

Name	Description
`spkrPinIndex`	The speaker pin for which delay is set.
`delay`	The delay setting on the path to the speaker pin in integral steps in the range `[spkr.minDelay,spkr.maxDelay]`.

Return Values
Standard.

Precondition
— `(spkr.suppSpeakerPins & spkrPinIndex) == spkrPinIndex`
— `spkr.minDelay <= delay <= spkr.maxDelay`

Action
— Modify `spkr.pSpeakers[spkrPinIndex].delay`

Postcondition
`spkr.pSpeakers[spkrPinIndex].delay == delay`

7.2.10.5.3.3 GetDelay

Signature
```
uhErrorCode_t GetDelay (
    [in] uhSpkr_SpkrPinIndex_t spkrPinIndex,
    [out] pUInt32 pDelay );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the current delay setting for a given speaker pin.

Parameters

Table 466

Name	Description
`spkrPinIndex`	The speaker pin for which the delay setting is queried.
`pDelay`	Output parameter specifying the current delay setting on the path to the speaker pin in integral steps in the range `[spkr.minDelay,spkr.maxDelay]`.

Return Values
Standard.

Precondition
— `(spkr.suppSpeakerPins & spkrPinIndex) == spkrPinIndex`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— `*pDelay == spkr.pSpeakers[spkrPinIndex].delay`

7.2.10.5.4 `uh`lSpkrBassRedir

Qualifiers
None.

Description
This interface is used to enable or disable bass redirection functionality. The bass redirection routing is selected by the implementation in best conformity with the speaker-set specified[15].

Interface ID
`uuid( 882AC8EE-4B37-4251-8A25-8939D5CEEFB8 )`

7.2.10.5.4.1 Enable

Signature
```
uhErrorCode_t Enable ( [in] Bool active );
```

Qualifiers
— synchronous

— single-threaded

Description
This function activates or deactivates bass redirection. When bass-redirection is activated, a best match is found between the speaker set and the input stream configuration.

Parameters

Table 467

Name	Description
`active`	Bass redirection activation set by the user. It is `True` when bass-redirection needs to be activated. It is `False` when bass-redirection needs to be deactivated

Return Values
Standard.

15 Refer to subclause 7.2.10.1.1.2 Bas Redirection.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Modify `spkr.currentBassRedirectionState`

Postcondition
`spkr.currentBassRedirectionState == active`

7.2.10.5.4.2 GetEnabled

Signature
```
uhErrorCode_t GetEnabled ( [out] pBool pActive );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the current bass redirection activation setting.

Parameters

Table 468

Name	Description
`pActive`	Output parameter indicating the current bass redirection activation setting.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pActive == spkr.currentBassRedirectionState`

7.2.10.5.4.3 SetSpkrType

Signature
```
uhErrorCode_t SetSpkrType (
    [in] uhSpkr_SpkrPinIndex_t spkrPinIndex,
    [in] uhSpkr_SpkrType_t spkrType );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— single-threaded

Description
This function sets the speaker type for a given speaker pin. The bass redirection routing is automatically decided by the LC based on the speaker pin type specification.

Parameters

Table 469

Name	Description
`spkrPinIndex`	The speaker pin for which the type is specified.
`spkrType`	The speaker type specification.

Return Values
Standard.

Precondition
```
(spkr.suppSpeakerPins & spkrPinIndex) == spkrPinIndex
```

Action
— Modify `spkr.pSpeakers[spkrPinIndex].speakerType`

Postcondition
```
spkr.pSpeakers[spkrPinIndex].speakerType == spkrType
```

7.2.10.5.4.4 GetSpkrType

Signature
```
uhErrorCode_t GetSpkrType (
    [in] uhSpkr_SpkrPinIndex_t spkrPinIndex,
    [out] puhSpkr_SpkrType_t pSpkrType );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the current speaker type for a given speaker pin.

Parameters

Table 470

Name	Description
`spkrPinIndex`	The speaker pin for which the speaker type is queried.
`pSpkrType`	Output parameter specifying the current speaker type.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
```
(spkr.suppSpeakerPins & spkrPinIndex) == spkrPinIndex
```

Action
None.

Postcondition
```
*pSpkrType == spkr.pSpeakers[spkrPinIndex].speakerType
```

7.2.10.5.4.5 GetStandardCompliancy

Signature
```
uhErrorCode_t GetStandardCompliancy (
    [out] puhSpkr_BassRedirStdCompliancy_t pStandard );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the standard to which the mapping between the speaker configuration and the routing of the bass signal is compliant, when bass redirection is activated.

Parameters

Table 471

Name	Description
`pStandard`	Output parameter returning the standard to which the mapping between the speaker configuration and the routing of the bass signal is compliant, when bass redirection is activated.

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
*pStandard == spkr.bassRedirStdCompliancy
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 Video interface suites

8.1 Broadcast Video decoding

8.1.1 UhlAnaAvDem

8.1.1.1 Concepts

8.1.1.1.1 Introduction

The function of an audio and video demodulator is to receive an IF, demodulate the signal, and output the resulting Y/CVBS video and encoded audio signal.

Apart from analog demodulation, the analog audio & video demodulator determines the TV system of the incoming signal so the sound carrier can be located.

8.1.1.2 Types & Constants

8.1.1.2.1 Public Types & Constants

8.1.1.2.1.1 Error Codes

None.

8.1.1.2.1.2 uhAnaAvDem_Notification_t

Signature

```
typedef enum _uhAnaAvDem_Notification_t {
    uhAnaAvDem_SubscriptionChanged  = 0x00000001,
    uhAnaAvDem_TvSystemFound        = 0x00000002,
    uhAnaAvDem_CarrierFound         = 0x00000004
} uhAnaAvDem_Notification_t, *puhAnaAvDem_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the notification functions in the interface `uhIAnaAvDemNtf`. They can be used to individually turn on and off notifications.

Values

Table 472

Name	Description
`uhAnaAvDem_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhAnaAvDem_TvSystemFound`	Represents `OnTvSystemDetected`.
`uhAnaAvDem_CarrierFound`	Represents `OnCarrierFound`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.1.2.1.3 uhAnaAvDem_Notifications_t

Signature
```
typedef Int32 uhAnaAvDem_Notifications_t, *puhAnaAvDem_Notifications_t;
```

Qualifiers
— enum-set

Description
Defines the set-type for `uhAnaAvDem_Notification_t`.

8.1.1.2.1.4 uhAnaAvDem_AllNotifications

Signature
```
const uhAnaAvDem_Notifications_t uhAnaAvDem_AllNotifications = 0x00000007;
```

Qualifiers
None.

Description
The set of all notifications of `uhAnaAvDem_Notification_t`.

8.1.1.2.1.5 uhAnaAvDem_TvSystem_t

Signature
```
typedef enum _uhAnaAvDem_TvSystem_t {
    uhAnaAvDem_TvSystemInvalid  = 0x80000000,
    uhAnaAvDem_TvSystemUnknown  = 0x40000000,
    uhAnaAvDem_TvSystemBG       = 0x00000001,
    uhAnaAvDem_TvSystemDK       = 0x00000002,
    uhAnaAvDem_TvSystemI        = 0x00000004,
    uhAnaAvDem_TvSystemH        = 0x00000008,
    uhAnaAvDem_TvSystemL        = 0x00000010,
    uhAnaAvDem_TvSystemL1       = 0x00000020,
    uhAnaAvDem_TvSystemM        = 0x00000040,
    uhAnaAvDem_TvSystemN        = 0x00000080
} uhAnaAvDem_TvSystem_t, *puhAnaAvDem_TvSystem_t;
```

Qualifiers
— enum-element

Description
Values of this type represent the possible TV systems for the platform. Not all these TV systems have to be supported.

Values

Table 473

Name	Description
`uhAnaAvDem_TvSystemInvalid`	The TV system property is temporarily not available.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhAnaAvDem_TvSystemUnknown	The TV system property is available, yet could not be determined.
uhAnaAvDem_TvSystemBG	TV system BG: 50 Hz, 625 lines, sound carrier at 5.5 MHz
uhAnaAvDem_TvSystemDK	TV system DK: 50 Hz, 625 lines, sound carrier at 6.5 MHz
uhAnaAvDem_TvSystemI	TV system I: 50 Hz, 625 lines, sound carrier at 6.0 MHz
uhAnaAvDem_TvSystemH	TV system H: 50 Hz, 625 lines, sound carrier at 5.5 MHz
uhAnaAvDem_TvSystemL	TV system L: 50 Hz, 625 lines, sound carrier at 6.5 MHz (positive modulated)
uhAnaAvDem_TvSystemL1	TV system L: 50 Hz, 625 lines, sound carrier at -6.5 MHz (positive modulated)
uhAnaAvDem_TvSystemM	TV system M: 60 Hz, 525 lines, sound carrier at 4.5 MHz
uhAnaAvDem_TvSystemN	TV system N: 50 Hz, 625 lines, sound carrier at 4.5 MHz

8.1.1.2.1.6 uhAnaAvDem_TvSystemSet_t

Signature

```
typedef Int32 uhAnaAvDem_TvSystemSet_t, *puhAnaAvDem_TvSystemSet_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhAnaAvDem_TvSystem_t.

8.1.1.2.1.7 uhAnaAvDem_Modulation_t

Signature

```
typedef enum _uhAnaAvDem_Modulation_t {
    uhAnaAvDem_ModulationUnknown   = 0,
    uhAnaAvDem_ModulationPositive  = 1,
    uhAnaAvDem_ModulationNegative  = 2
} uhAnaAvDem_Modulation_t, *puhAnaAvDem_Modulation_t;
```

Qualifiers

— enum

Description

Values of this type represent the supported analog modulation types for the platform.

Values

Table 474

Name	Description
uhAnaAvDem_ModulationUnknown	The modulation type is not known.
uhAnaAvDem_ModulationPositive	Positive modulation.
uhAnaAvDem_ModulationNegative	Negative modulation.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.1.2.2 Model Types & Constants

None.

8.1.1.3 Logical Component

8.1.1.3.1 Interface-Role Model

Figure 62 depicts the interface-role model of the analog audio & video demodulator logical component. It shows the control and notification interfaces and the roles involved with this logical component.

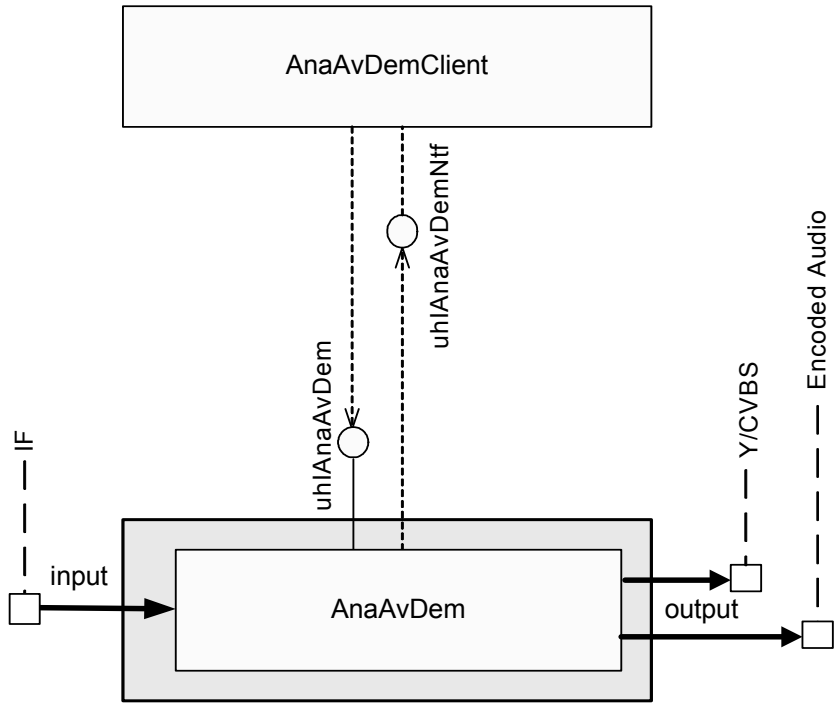


Figure 62 — Interface Role Model

The grey box in Figure 62 denotes an analogue audio & video demodulator logical component containing the `AnaAvDem` role providing one mandatory (solid line) control interfaces to its clients.

Clients can optionally provide call-back interfaces allowing them to receive notifications of events in the `AnaAvDem`.

8.1.1.3.2 Diversity

8.1.1.3.2.1 Provided Interfaces

Table 475

Role	Interface	Presence
`AnaAvDem`	`uhIAnaAvDem`	mandatory
`AnaAvDem`	`uhIAnaAvDemNtf`	optional

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.1.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of the logical component. For a description of the attributes see Roles subclause, and for how they are initialized see subclause 8.1.1.3.3.

Table 476

Role	Attribute
AnaAvDem	suppTvSystems

8.1.1.3.2.3 Constraints

The set of supported TV systems tun.suppTvSystems does not include uhAnaAvDem_TvSystemUnknown or uhAnaAvDem_TvSystemInvalid but only existing TV systems from the range [uhAnaAvDem_TvSystemBG..uhAnaAvDem_TvSystemN]. See the *Invariants* of AnaAvDem role (subclause 8.1.1.4) for the complete constraints.

8.1.1.3.3 Instantiation

8.1.1.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 477

Type	Object	Multiplicity
AnaAvDem	Aavd	1

8.1.1.3.3.2 Initial State

```
Aavd.setTvSystem == uhAnaAvDem_TvSystemUnknown

Aavd.actualTvSystem == uhAnaAvDem_TvSystemUnknown

Aavd.currentModulation == uhAnaAvDem_ModulationUnknown

Aavd.carrierPresent == False
```

8.1.1.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.1.4 Roles

8.1.1.4.1 AnaAvDem

Signature

```
role AnaAvDem {
    const uhAnaAvDem_TvSystemSet_t suppTvSystems;
    uhAnaAvDem_TvSystem_t          setTvSystem;
    uhAnaAvDem_TvSystem_t          actualTvSystem;
    uhAnaAvDem_Modulation_t        currentModulation;
    Bool                           carrierPresent;
}
```

Qualifiers

— root

Description

An `AnaAvDem` controls the analog demodulation attributes.

Independent Attributes

Table 478

Name	Description
`suppTvSystems`	The supported TV systems.
`setTvSystem`	The TV system as it is set via the `SetTvSystem` function.
`actualTvSystem`	The detected TV system.
`currentModulation`	The current modulation.
`carrierPresent`	The presence (absence) of a valid signal

Invariants

```
setTvSystem in suppTvSystems or
actualTvSystem == uhAnaAvDem_TvSystemUnknown

actualTvSystem in suppTvSystems or
actualTvSystem == uhAnaAvDem_TvSystemUnknown or
actualTvSystem == uhAnaAvDem_TvSystemInvalid
```

Instantiation

A `AnaAvDem` role is always instantiated as part of an analog audio & video demodulator logical component; see subclause 8.1.1.3.3.

Streaming Behavior

The `AnaAvDem` role tunes the incoming IF signal into a demodulated signal that can be handled by analog audio and video decoder.

Active Behavior

The `AnaAvDem` has standard notification behavior (see `uhIAnaAvDemNtf`) except for the following event:

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 479

Event	Action
A TV system is detected on the input after automatic detection was requested.	— Modify `actualTvSystem` — Notify `OnTvSystemDetected`(True, `actualTvSystem`)
The TV system detection has been aborted via a new `SetTvSystem` function call	— Modify setTvSystem — Modify actualTvSystem — Notify `OnTvSystemDetected` (False, actualTvSystem)
A signal (carrier) has been found after a new frequency has been set into the tuner	— Modify `carrierPresent` — Notify `OnCarrierFound`(cookie, True);

8.1.1.4.2 AnaAvDemClient

Signature
role `AnaAvDemClient {}`

Qualifiers
— actor

Description
A `AnaAvDemClient` role represents a user of the control interfaces offered by the `AnaAvDem` role. It can optionally provide the notification interfaces to receive notifications of events generated by the `AnaAvDem` role. No constraints apply to the behavior of the `AnaAvDemClient` role other than those implied by Figure 62.

8.1.1.5 Interfaces

8.1.1.5.1 uhlAnaAvDem

Qualifiers
None.

Description
This interface of the `AnaAvDem` role provides control of the demodulation parameters.

Interface ID
`uuid( 6A14206F-794E-4224-A139-180FCFB2F32F )`
Remarks

— In the following function specifications the name `Aavd` will be used to refer to the provider of this interface.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.1.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIAnaAvDemNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAnaAvDem_Notifications_t notifs );
```

Qualifiers

— subscribe-function

8.1.1.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIAnaAvDemNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAnaAvDem_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.1.5.1.3 GetSuppTvSystems

Signature

```
uhErrorCode_t GetSuppTvSystems ( [out] puhAnaAvDem_TvSystemSet_t pSystems );
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns the TV systems that are supported.

Parameters

Table 480

Name	Description
`pSystems`	Output parameter returning the TV systems that are supported.

Return Values

Standard.

Precondition

`True`

Action

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pSystems == Aavd.suppTvSystems`

8.1.1.5.1.4 SetTvSystem

Signature
```
uhErrorCode_t SetTvSystem ( [in] uhAnaAvDem_TvSystem_t system );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the TV system into the hardware. `Aavd.setTvSystem` and `Aavd.actualTvSystem` are changed to the new value. Any current TV system detection is interrupted.

Parameters

Table 481

Name	Description
`system`	Input parameter indicating the new TV system.

Return Values
Standard.

Precondition
`system in Aavd.suppTvSystems`

Action
— Modify `Aavd.setTvSystem`
— Modify `Aavd.actualTvSystem`

Postcondition
— `Aavd.setTvSystem == system`
— `Aavd.actualTvSystem == system`

8.1.1.5.1.5 DetectTvSystem

Signature
```
uhErrorCode_t DetectTvSystem ( );
```

Qualifiers
— asynchronous

Description
This function starts the TV system detection. After the detection finishes the notification OnTvSystemDetected is generated.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`carrierPresent == True`

Action
— Modify `actualTvSystem`

Postcondition
`actualTvSystem == uhAnaAvDem_TvSystemUnknown`

Asynchronous Action
— Modify `actualTvSystem`

Asynchronous Postcondition
`True`

8.1.1.5.1.6 GetTvSystem

Signature
```
uhErrorCode_t GetTvSystem ( [out] puhAnaAvDem_TvSystem_t pSystem );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the TV system as it was set via the `SetTvSystem` function.

Parameters

Table 482

Name	Description
`pSystem`	Output parameter returning the TV system that was set.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pSystem == Aavd.setTvSystem`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.1.5.1.7 GetActualTvSystem

Signature
```
uhErrorCode_t GetActualTvSystem ( [out] puhAnaAvDem_TvSystem_t pSystem );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the current TV system.

Parameters

Table 483

Name	Description
`pSystem`	Output parameter returning the current TV system.

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
*pSystem == Aavd.actualTvSystem
```

8.1.1.5.1.8 SetModulation

Signature
```
uhErrorCode_t SetModulation ( [in] uhAnaAvDem_Modulation_t modulation );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the modulation to positive or negative.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 484

Name	Description
`modulation`	Input parameter indicating positive or negative modulation.

Return Values
Standard.

Precondition
`modulation ≠ uhAnaAvDem_ModulationUnknown`

Action
— Modify `Aavd.currentModulation`

Postcondition
`Aavd.currentModulation == modulation`

Remarks
— Often, the modulation can not be detected by the underlying hardware and must be set to the right type before any other preset related action is done.

8.1.1.5.1.9 GetModulation

Signature
```
uhErrorCode_t GetModulation ( [out] puhAnaAvDem_Modulation_t pModulation );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the modulation as it was set via the `SetModulation` function.

Parameters

Table 485

Name	Description
`pModulation`	Output parameter returning the modulation that was set.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
`*pModulation == Aavd.currentModulation`

8.1.1.5.1.10 GetCarrierPresence

Signature
`uhErrorCode_t GetCarrierPresence ( [out] pBool pCarrier );`

Qualifiers
— synchronous

Description
The function returens the presence of a signal carrier.

Parameters

Table 486

Name	Description
`pCarrier`	The presence (`True`) or absence (`False`) of a signal

Precondition
`True`

Postcondition
`*pCarrier == carrierPresent`

8.1.1.5.2 uhIAnaAvDemNtf

Qualifiers
None.

Description
This interface can be provided by each instance of an `AnaAvDemClient` role that wants to receive notifications of events that occur in the `AnaAvDem`. The `Subscribe` function of `uhIAnaAvDem` can be used to subscribe to these events.

Interface ID
`uuid( 2F5B3F55-FDD7-43b1-BE36-E002E992969F )`

8.1.1.5.2.1 OnSubscriptionChanged

Signature
`[async]` `Void OnSubscriptionChanged (`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32 cookie,
    [in] uhAnaAvDem_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.1.5.2.2 OnTvSystemDetected

Signature
```
[async] Void OnTvSystemDetected (
    [in] UInt32 cookie,
    [in] Bool result,
    [in] uhAnaAvDem_TvSystem_t tvSystem );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the detection of a new TV system is finished or aborted.

Parameters

Table 487

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`result`	The detection did find a supported TV system (`True`) or the detection did not succeed or the detection was aborted (`False`).
`tvSystem`	The actual TV system.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.1.5.2.3 OnCarrierFound

Signature
```
[async] Void OnCarrierFound (
    [in] UInt32 cookie,
    [in] Bool carrier );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function notifies subscribed clients that a transition from no carrier to a carrier has been detected on the input. After this notification the client can start detection of signal properties like color system and TV system.

Parameters

Table 488

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`carrier`	The presence (`True`) or absence (`False`) of the carrier.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.2 UhlAnaVdec

8.1.2.1 Concepts

8.1.2.1.1 Introduction

Video signals are a way of transferring visual information from one point to another. The information may be from a VCR, channel broadcast, DVD player, game computer, or many other sources. The receiving device is usually a display like a television or monitor. On the sources side the signal is encoded, on the receiving side the signal is decoded first in order to display it. In case the signal is analog, the decoding can be done by an AVD. There are many ways of encoding the video data into an analog video signal, so the input of the AVD may be varied. Its output is always a video signal according the YUV video format, which is suitable for further processing in the video path. The AVD supplies information about the video signal to the AVD client and can be controlled by AVD client by means of its interfaces. The concepts relevant to understanding the AVD will be explained in the remainder of this subclause.

8.1.2.1.2 Video Coding Formats

8.1.2.1.2.1 Color data

Basically, the information in the video signal has to tell what (color data) and when (video timing) to display. The color can be expressed completely by using three orthogonal color components, setting up a color space.

Used color spaces for video coding are:

— RGB: Color space of basic colors red, green, and blue.

— YUV: Twisted and scaled RGB space, such that one component contains the luminance (black and white) information, U and V the chrominance (color type) information.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— YPbPr: Scaled version of the YUV color space.

— YCbCr: Digital version of the YPbPr color space, not detectable for an AVD (mentioned only for completeness).

The highest quality of video is obtained when the three components of the color space are transmitted separately, so using three signals. Video coding formats using three signals for color are called component video formats or CVI formats. They are named according the color space used, so RGB, YUV, or YPbPr format. However, three separate signals take a lot of bandwidth, which was especially a problem in the early days of video. Therefore, a so-called composite format, CVBS, was defined, where all color data is contained in one signal. In this case the AVD has to separate the several components upon reception of the data. Because the different components in one signal may interfere with one another, the picture quality of a composite video signal is often less than that of a component video signal. An improvement of the CVBS format is the Y/C format, where two signals are used for transmitting the color data, one for the luminance (Y) and one for the chrominance (C). This is a format based on the YUV color space, where the U and V component are still together in a composite signal. Because the Y/C format is still partly composite, we will treat it in this specification as a composite video format.

8.1.2.1.2.2 Video Timing

Of course, the color data only cannot specify a moving picture. Therefore, it has to be specified when the color is to be displayed. The notion of a moving picture is obtained by transmitting a series of still images. The start of a new image is indicated by adding timing information to the video signal, called vertical sync. The image in turn is composed of several scan lines, transmitted from top to bottom, where each start of a new line is indicated in the video signal with horizontal sync information. The sync information may be transmitted with the color data, like in the CVBS format, or in separate signal(s), like in the RGB format.

8.1.2.1.2.3 Interlaced versus progressive scan

In order to save bandwidth, a technique has been defined called interlacing. Here the images contain alternately just the odd or just the even lines. On the other hand, when each image contains all lines, the technique is called progressive scan or progressive in short. A full image, containing all lines, is called a frame, and an image composed of just the even or odd lines is called a field. By definition, two fields make up a frame. The speed at which the images are sent is called the vertical frequency or field frequency (which is actually the frame frequency for progressive). Typical field frequencies used for television are 50Hz or 60Hz, meaning 50 or 60 fields per second. Note that with displaying a 50Hz progressive signal twice as much information is processed per second as with displaying a 50Hz interlaced signal. This becomes clear when defining the horizontal frequency or line frequency: this is the field frequency multiplied with the number of lines per field. Since the progressive field contains twice as many lines as the interlaced field, the horizontal frequency is also twice as high with progressive. Typically, the number of lines per field is 525 or 625 for progressive and 262.5 or 312.5 for interlaced. So for example, the horizontal frequency for a 50Hz progressive signal with 625 lines per field is 31250 Hz. In order to express the horizontal frequency independent of the exact number of lines and field frequency, it is also written as 1Fh for interlaced and 2Fh for progressive. An interlaced signal can also be 2Fh, if each field is encoded twice in the video signal, doubling the field frequency. In the same way, even 3Fh signals exist. Finally, there exists a signal, which is produced by some game computers and is called non-interlaced. This is much like a normal interlaced signal, where all fields just contain the odd lines, or all fields contain the even lines. Like with a normal interlaced signal the horizontal frequency is 1Fh. The AVD considers this kind of signal to be interlaced.

8.1.2.1.2.4 Video properties

The video properties are information about the video signal that may be useful to know for the AVD client. Most of the properties tell something about the video timing.

They are:

— Video presence: indicates whether a video signal is present at all. Usually this is derived from the detection of sync and color system.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Video coding type: one of the types as defined in subclause 8.1.2.1.2.1.

— Interlaced mode: indicates whether the mode is interlaced or progressive, see subclause 8.1.2.1.2.3.

— Field frequency: see subclause 8.1.2.1.2.3.

— Number of frame lines: see subclause 8.1.2.1.2.3.

— VCR signal presence: indicates whether the video signal is coming from a VCR or not. The AVD is able to detect this because such signal often suffers from so called phase jumps. This means that the VCR tape is stretched, and consequently the sync is not in place anymore. Maybe for this reason, maybe for another reason, it can be useful for the AVD client to know that the signal is from a VCR.

— Copy protection: Indicates that a copy-protected signal has been detected, that is, the signal may be played but not be provided for recording on any of the outputs.

8.1.2.1.2.5 Comb filter

A comb filter can separate the luminance and chrominance component of a CVBS video signal, improving the resulting picture quality. The filter is named a comb filter because pieces of the luminance and chrominance signal appear alternately in the CVBS video signal. So in order to filter one of them the filter needs a comb-like shape. The comb filter is only applicable in case of a video signal according the PAL or NTSC color system standard, SECAM signals cannot be “combed”. A Y/C coded signal and component video signals do not need to be “combed”, because the luminance and chrominance are already separated in those signals. The AVD client can enable or disable the comb filter of the AVD. Usually it will be desirable to enable the comb filter because (in most cases) it improves the picture quality. In some cases however, this improvement may be undesirable. For example, when the picture is displayed in two windows at the same time (double window) and the picture in one window can be improved with the comb filter, but not in the other window. This would show annoying quality differences between the two pictures. For that reason the client would like to disable the comb filter for the window that can be “combed”.

8.1.2.1.2.6 Hue

Technically, hue is defined as the wavelength of the color. It is the type of color, independent of the luminance. Thus, it is specified by the angle (or phase) between the U and V component when taking the YUV color space as a basis. As the U and V information is combined in composite video signals, phase errors may occur with encoding or decoding, yielding a wrong hue. The AVD offers means to adjust the hue in order to compensate for phase errors. Usually this function is only relevant for video signals according the NTSC color system standard, because in PAL or SECAM signals phase errors are cancelled out by definition.

8.1.2.1.3 Color systems

Historically, three basic color systems for transmitting color TV signals have been developed: NTSC, PAL and SECAM. All three color systems use a composite video format, based on the YUV color space. The luminance information is transmitted on the so-called carrier frequency, the chrominance information on the sub-carrier frequency or frequencies. Several variants of the basic color systems exist, indicated by a letter like B, M, K, etc. These letters refer to the way the audio is encoded in the signal and are as such beyond the scope of this specification. However, in some of these variants, the color information is on a different sub-carrier frequency also making them also relevant for this specification. We will describe all combinations of basic color system and sub-carrier frequency in the remainder of this paragraph.

8.1.2.1.3.1 NTSC

NTSC was the first color system to be developed for television in the United States of America. Several variants exist, but all have a field frequency of 60 Hz and a number of frame lines of 525. The color sub-carrier frequency is located at 3.579545 or at 4.433619 MHz. The last variant never occurs in broadcasts anymore, it is only transmitted by some multi-standard VCR’s.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.1.3.2 PAL

PAL was developed in Europe to be an improvement of the NTSC color system, with respect to the hue errors. In PAL signals, the phase of the V component is reversed every line, producing complementary errors. This enables the receiver to average the phase errors and cancel them out. This is the reason why hue control is not needed with PAL signals. PAL color systems have a field frequency of 50 or 60 Hz, and a number of frame lines of 525 or 625. The color sub-carrier frequency can be at 3.575611, 3.582056, or 4.433619 MHz.

8.1.2.1.3.3 SECAM

SECAM was developed in France, and like PAL, to be an improvement of the NTSC color system with respect to the hue errors. This is achieved by sending the U and V information each alternating line. By comparing two consecutive fields the receiver can cancel out the phase errors, so also SECAM signals need no hue correction. All SECAM variants have a field frequency of 60 Hz and transmit 625 lines per frame. The SECAM color system standard sends the U and V information on two different sub-carrier frequencies, being 4.250000 and 4.406250 MHz respectively.

8.1.2.2 Types & Constants

8.1.2.2.1 Public Types & Constants

8.1.2.2.1.1 Error Codes

None.

8.1.2.2.1.2 uhAnaVdec_ValStat_t

Signature

```
typedef enum _uhAnaVdec_ValStat_t {
    uhAnaVdec_ValStatInvalid = -2,
    uhAnaVdec_ValStatUnknown = -1,
    uhAnaVdec_ValStatKnown   =  1
} uhAnaVdec_ValStat_t, *puhAnaVdec_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 489

Name	Description
uhAnaVdec_ValStatInvalid	The property/properties are temporarily not available.
uhAnaVdec_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhAnaVdec_ValStatKnown	The property/properties are available and determined.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.2.1.3 uhAnaVdec_VideoCoding_t

Signature

```
typedef enum _uhAnaVdec_VideoCoding_t {
    uhAnaVdec_VideoCodingInvalid = -2,
    uhAnaVdec_VideoCodingUnknown = -1,
    uhAnaVdec_VideoCodingCVBS    =  1,
    uhAnaVdec_VideoCodingYC      =  2,
    uhAnaVdec_VideoCodingYUV     =  3,
    uhAnaVdec_VideoCodingYPbPr   =  4,
    uhAnaVdec_VideoCodingRGB     =  5
} uhAnaVdec_VideoCoding_t, *puhAnaVdec_VideoCoding_t;
```

Qualifiers
None.

Description
Values of this type describe the video coding used for the analogue video signal input.

Values

Table 490

Name	Description
uhAnaVdec_VideoCodingInvalid	The video coding property is temporarily not available.
uhAnaVdec_VideoCodingUnknown	The video coding property is available, yet it could not be determined.
uhAnaVdec_VideoCodingCVBS	CVBS video coding.
uhAnaVdec_VideoCodingYC	Y/C video coding.
uhAnaVdec_VideoCodingYUV	YUV video coding.
uhAnaVdec_VideoCodingYPbPr	YPbPr video coding.
uhAnaVdec_VideoCodingRGB	RGB video coding.

8.1.2.2.1.4 uhAnaVdec_ScanType_t

Signature

```
typedef enum _uhAnaVdec_ScanType_t {
    uhAnaVdec_ScanTypeInvalid     = -2,
    uhAnaVdec_ScanTypeUnknown     = -1,
    uhAnaVdec_ScanTypeInterlaced  =  1,
    uhAnaVdec_ScanTypeProgressive =  2
} uhAnaVdec_ScanType_t, *puhAnaVdec_ScanType_t;
```

Qualifiers
None.

Description
Values of this type describe the way the field(s) of the analogue video signal input are constructed into frames.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 491

Name	Description
uhAnaVdec_ScanTypeInvalid	The scan type property is temporarily not available.
uhAnaVdec_ScanTypeUnknown	The scan type property is available, yet it could not be determined.
uhAnaVdec_ScanTypeInterlaced	Each frame consists of two fields.
uhAnaVdec_ScanTypeProgressive	Each frame consists of one field.

8.1.2.2.1.5 uhAnaVdec_ColorSystem_t

Signature

```
typedef enum _uhAnaVdec_ColorSystem_t {
    uhAnaVdec_ColorSystemInvalid   = 0x80000000,
    uhAnaVdec_ColorSystemUnknown   = 0x40000000,
    uhAnaVdec_ColorSystemAuto      = 0x00000001,
    uhAnaVdec_ColorSystemPal       = 0x00000002,
    uhAnaVdec_ColorSystemSecam     = 0x00000004,
    uhAnaVdec_ColorSystemNtsc      = 0x00000008
} uhAnaVdec_ColorSystem_t, *puhAnaVdec_ColorSystem_t;
```

Qualifiers

— enum-element

Description

Values of this type describe in which way the color information of the analogue video signal input is coded.

Values

Table 492

Name	Description
uhAnaVdec_ColorSystemInvalid	The color system property is temporarily not available.
uhAnaVdec_ColorSystemUnknown	The color system property is available, yet it could not be determined.
uhAnaVdec_ColorSystemAuto	The color system will be determined automatically and changes will be notified.
uhAnaVdec_ColorSystemPal	Color system PAL.
uhAnaVdec_ColorSystemSecam	Color system SECAM.
uhAnaVdec_ColorSystemNtsc	Color system NTSC.

8.1.2.2.1.6 uhAnaVdec_ColorSystemSet_t

Signature

```
typedef Int32 uhAnaVdec_ColorSystemSet_t, *puhAnaVdec_ColorSystemSet_t;
```

Qualifiers

— enum-set

Description

This type represents sets of type `uhAnaVdec_ColorSystem_t`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.2.1.7 uhAnaVdec_LineFreq_t

Signature

```
typedef enum _uhAnaVdec_LineFreq_t {
    uhAnaVdec_LineFreqInvalid = -2,
    uhAnaVdec_LineFreqUnknown = -1,
    uhAnaVdec_LineFreq1Fh     =  1,
    uhAnaVdec_LineFreq2Fh     =  2,
    uhAnaVdec_LineFreq3Fh     =  3
} uhAnaVdec_LineFreq_t, *puhAnaVdec_LineFreq_t;
```

Qualifiers
None.

Description
Values of this type describe the horizontal or line frequency of the analogue video signal input.

Values

Table 493

Name	Description
uhAnaVdec_LineFreqInvalid	The line frequency property is temporarily not available.
uhAnaVdec_LineFreqUnknown	The line frequency property is available, yet it could not be determined.
uhAnaVdec_LineFreq1Fh	1Fh video input signal.
uhAnaVdec_LineFreq2Fh	2Fh video input signal.
uhAnaVdec_LineFreq3Fh	3Fh video input signal.

8.1.2.2.1.8 uhAnaVdec_Protection_t

Signature

```
typedef enum _uhAnaVdec_Protection_t {
    uhAnaVdec_ProtectionInvalid       = 0x80000000,
    uhAnaVdec_ProtectionUnknown       = 0x40000000,
    uhAnaVdec_ProtectionNone          = 0x00000001,
    uhAnaVdec_ProtectionCopy          = 0x00000002,
    uhAnaVdec_ProtectionScrambled     = 0x00000004,
    uhAnaVdec_ProtectionMacro         = 0x00000008,
    uhAnaVdec_ProtectionMacroType1    = 0x00000010,
    uhAnaVdec_ProtectionMacroType2    = 0x00000020,
    uhAnaVdec_ProtectionMacroType3    = 0x00000040
} uhAnaVdec_Protection_t, *puhAnaVdec_Protection_t;
```

Qualifiers
— enum-element

Description
The different protection mechanisms that can be determined. The values of this type are OR-able. If a copy protection is present this should always be indicated by `uhAnaVdec_ProtectionCopy`. Next to that, the content can also be scrambled, which should be indicated by `uhAnaVdec_ProtectionScrambled`. If the type of copy protection is Macrovision, this should also be indicated by `uhAnaVdec_ProtectionMacro`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

together with an indication of the exact type of Macrovision used (if this information is available/could be determined).

Values

Table 494

Name	Description
uhAnaVdec_ProtectionInvalid	The protection property is temporarily not available.
uhAnaVdec_ProtectionUnknown	The protection property is available, yet it could not be determined.
uhAnaVdec_ProtectionNone	The signal is not protected
uhAnaVdec_ProtectionCopy	Indicates that the video signal is copy-protected.
uhAnaVdec_ProtectionScrambled	Indicates that the video signal is protection by scrambling, e.g. the signal has to be de-scrambled before it can be rendered.
uhAnaVdec_ProtectionMacro	Indicates that the video signal is protected using Macrovision.
uhAnaVdec_ProtectionMacroType1	Indicates that the video signal is protected using Macrovision Type 1: AGC pulses/pseudo syncs present.
uhAnaVdec_ProtectionMacroType2	Indicates that the video signal is protected using Macrovision Type 2: AGC pulses/pseudo syncs and 2-line color striping present.
uhAnaVdec_ProtectionMacroType3	Indicates that the video signal is protected using Macrovision Type 3: AGC pulses/pseudo syncs and 4-line color striping present.

8.1.2.2.1.9 uhAnaVdec_ColorFreq_t

Signature

```
typedef enum _uhAnaVdec_ColorFreq_t {
    uhAnaVdec_ColorFreqInvalid     = 0x80000000,
    uhAnaVdec_ColorFreqUnknown     = 0x40000000,
    uhAnaVdec_ColorFreqAuto        = 0x00000001,
    uhAnaVdec_ColorFreq3575611     = 0x00000002,
    uhAnaVdec_ColorFreq3579545     = 0x00000004,
    uhAnaVdec_ColorFreq3582056     = 0x00000008,
    uhAnaVdec_ColorFreq4250000     = 0x00000010,
    uhAnaVdec_ColorFreq4406250     = 0x00000020,
    uhAnaVdec_ColorFreq4433619     = 0x00000040
} uhAnaVdec_ColorFreq_t, *puhAnaVdec_ColorFreq_t;
```

Qualifiers

— enum-element

Description

Values of this type describe which sub-carrier frequencies are used for the known color systems. The values are OR-able because in case of SECAM there are two color sub-carrier frequencies used, instead of one as with PAL and NTSC. The SECAM color sub-carrier frequencies can thus be indicated by OR-ing both frequencies.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 495

Name	Description
uhAnaVdec_ColorFreqInvalid	The sub-carrier frequency is temporarily not available.
uhAnaVdec_ColorFreqUnknown	The sub-carrier frequency is available, yet it could not be determined.
uhAnaVdec_ColorFreqAuto	The sub-carrier frequency is determined automatically.
uhAnaVdec_ColorFreq3575611	Represents a sub-carrier frequency of 3575611 Hz (PAL).
uhAnaVdec_ColorFreq3579545	Represents a sub-carrier frequency of 3579545 Hz (NTSC).
uhAnaVdec_ColorFreq3582056	Represents a sub-carrier frequency of 3582056 Hz (PAL).
uhAnaVdec_ColorFreq4250000	Represents a sub-carrier frequency of 4250000 Hz (SECAM).
uhAnaVdec_ColorFreq4406250	Represents a sub-carrier frequency of 4406250 Hz (SECAM)
uhAnaVdec_ColorFreq4433619	Represents a sub-carrier frequency of 4433619 Hz (PAL and NTSC)

8.1.2.2.1.10 uhAnaVdec_ColorFreqSet_t

Signature

```
typedef Int32 uhAnaVdec_ColorFreqSet_t, *puhAnaVdec_ColorFreqSet_t;
```

Qualifiers

— enum-set

Description

This type represents sets of type `uhAnaVdec_ColorFreq_t`.

8.1.2.2.1.11 uhAnaVdec_Notification_t

Signature

```
typedef enum _uhAnaVdec_Notification_t {
    uhAnaVdec_SubscriptionChanged          = 0x00000001,
    uhAnaVdec_VideoPresenceChanged         = 0x00000002,
    uhAnaVdec_VideoCodingChanged           = 0x00000004,
    uhAnaVdec_LineFrequencyChanged         = 0x00000008,
    uhAnaVdec_VideoTimingChanged           = 0x00000010,
    uhAnaVdec_VcrSignalPresenceChanged     = 0x00000020,
    uhAnaVdec_ProtectionChanged            = 0x00000040,
    uhAnaVdec_PixelsPerLineChanged         = 0x00000080,
    uhAnaVdec_StableOutput                 = 0x00000100
} uhAnaVdec_Notification_t, *puhAnaVdec_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the notification functions in the interface `uhIAnaVdecNtf`. They can be used to individually turn on and off notifications.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 496

Name	Description
uhAnaVdec_SubscriptionChanged	Represents OnSubscriptionChanged
uhAnaVdec_VideoPresenceChanged	Represents OnVideoPresenceChanged.
uhAnaVdec_VideoCodingChanged	Represents OnVideoCodingChanged.
uhAnaVdec_LineFrequencyChanged	Represents OnLineFrequencyChanged.
uhAnaVdec_VideoTimingChanged	Represents OnVideoTimingChanged.
uhAnaVdec_VcrSignalPresenceChanged	Represents OnVcrSignalPresenceChanged
uhAnaVdec_ProtectionChanged	Represents OnProtectionChanged
uhAnaVdec_PixelsPerLineChanged	Represents OnPixelsPerLineChanged
uhAnaVdec_StableOutput	Represents OnStableOutput.

8.1.2.2.1.12 uhAnaVdec_Notifications_t

Signature

```
typedef Int32 uhAnaVdec_Notifications_t, *puhAnaVdec_Notifications_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhAnaVdec_Notification_t.

8.1.2.2.1.13 uhAnaVdec_AllNotifications

Signature

```
const uhAnaVdec_Notifications_t uhAnaVdec_AllNotifications = 0x000001FF;
```

Qualifiers

None.

Description

The set of all notifications of uhIAnaVdecNtf. Defines the set of all values of type uhAnaVdec_Notifications_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.1.2.2.1.14 uhAnaVdec_CombFilterNotification_t

Signature

```
typedef enum _uhAnaVdec_CombFilterNotification_t {
    uhAnaVdec_CombFilterSubscriptionChanged = 0x00000001,
    uhAnaVdec_CombFilterChanged             = 0x00000002
} uhAnaVdec_CombFilterNotification_t, *puhAnaVdec_CombFilterNotification_t;
```

Qualifiers

— enum-element

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent the notification functions in the interface `uhIAnaVdecCombFilterNtf`. They can be used to individually turn on and off notifications.

Values

Table 497

Name	Description
`uhAnaVdec_CombFilterSubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhAnaVdec_CombFilterChanged`	Represents `OnCombFilterChanged`.

8.1.2.2.1.15 uhAnaVdec_CombFilterNotifications_t

Signature
```
typedef Int32 uhAnaVdec_CombFilterNotifications_t, *puhAnaVdec_CombFilterNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions, which are typically constructed by logical OR-ing of values of type uhAnaVdec`_CombFilterNotification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.1.2.2.1.16 uhAnaVdec_CombFilterAllNotifications

Signature
```
const uhAnaVdec_CombFilterNotifications_t uhAnaVdec_CombFilterAllNotifications = 0x00000003;
```

Qualifiers
None.

Description
Defines the set of all values of type uhAnaVdec`_CombFilterNotification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.1.2.2.1.17 uhAnaVdec_ColDecNotification_t

Signature
```
typedef enum _uhAnaVdec_ColDecNotification_t {
    uhAnaVdec_ColDecSubscriptionChanged = 0x00000001,
    uhAnaVdec_ColorSystemChanged        = 0x00000002
} uhAnaVdec_ColDecNotification_t, *puhAnaVdec_ColDecNotification_t;
```

Qualifiers
— enum-element

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent the notification functions in the interface `uhIAnaVdecColDecNtf`. They can be used to individually turn on and off notifications.

Values

Table 498

Name	Description
`uhAnaVdec_ColDecSubscriptionChanged`	Represents OnSubscriptionChanged.
`uhAnaVdec_ColorSystemChanged`	Represents `OnColorSystemChanged`.

8.1.2.2.1.18 uhAnaVdec_ColDecNotifications_t

Signature
```
typedef Int32 uhAnaVdec_ColDecNotifications_t,
    *puhAnaVdec_ColDecNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions, which are typically constructed by logical OR-ing of values of type uhAnaVdec`_ColDecNotification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.1.2.2.1.19 uhAnaVdec_ColDecAllNotifications

Signature
```
const uhAnaVdec_ColDecNotifications_t
    uhAnaVdec_ColDecAllNotifications = 0x00000003;
```

Qualifiers
None.

Description
Defines the set of all values of type uhAnaVdec`_ColDecNotification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.1.2.2.2 Model Types & Constants

None.

8.1.2.3 Logical Component

8.1.2.3.1 Interface-Role Model

Fugure 59 shows the interface-role model of the AVD. It shows the interfaces, roles, attributes, and input and output streams used in the specification of the AVD.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

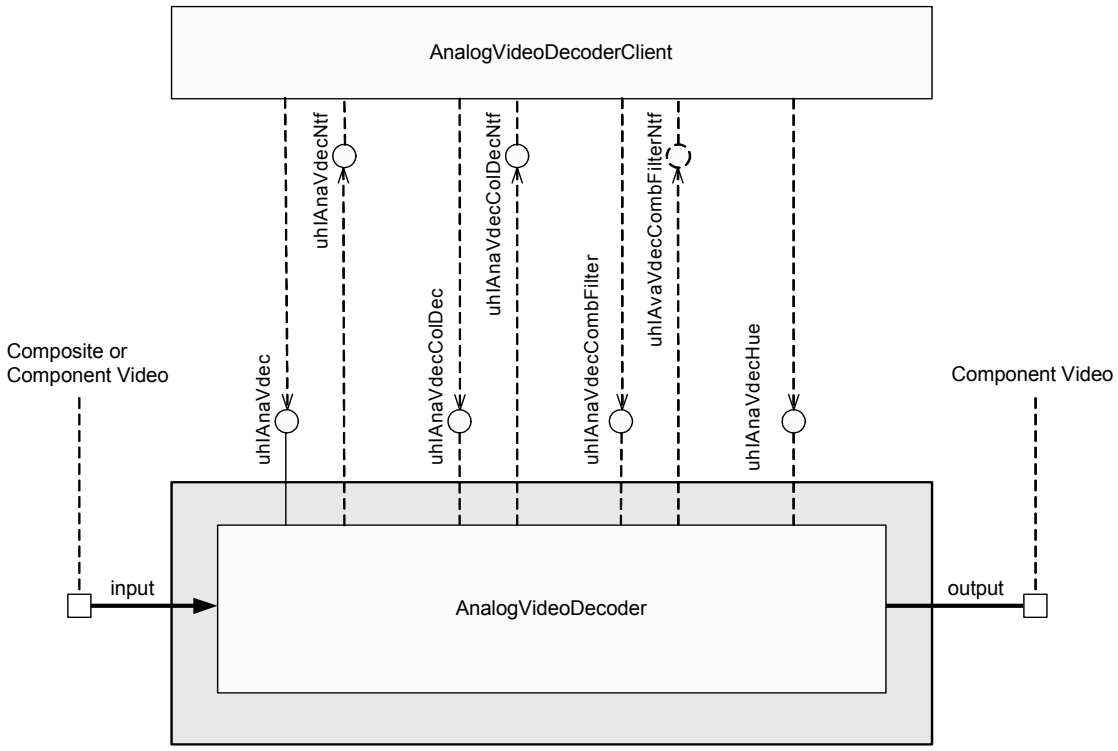


Figure 63 — Interface-Role Model of the Analog Video Decoder

8.1.2.3.2 Diversity

8.1.2.3.2.1 Provided Interfaces

The following table defines which interfaces are optional and which ones are mandatory:

Table 499

Role	Interface	Presence
`AnalogVideoDecoder`	`uhIAnaVdec`	Mandatory
`AnalogVideoDecoderClient`	`uhIAnaVdecNtf`	Optional
`AnalogVideoDecoder`	`uhIAnaVdecColDec`	Optional
`AnalogVideoDecoderClient`	`uhIAnaVdecColDecNtf`	Optional
`AnalogVideoDecoder`	`uhIAnaVdecCombFilter`	Optional
`AnalogVideoDecoderClient`	`uhIAnaVdecCombFilterNtf`	Optional
`AnalogVideoDecoder`	`uhIAnaVdecHue`	Optional

8.1.2.3.2.2 Configurable Items

The configurable items of an AVD are represented by the following attributes, which are set at creation time of an AVD:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 500

Role	Attribute
AnalogVideoDecoder	minHue
AnalogVideoDecoder	maxHue
AnalogVideoDecoder	suppColorSystems
AnalogVideoDecoder	suppColorFrequencies

8.1.2.3.2.3 Constraints

— uhAnaVdec_ColorSystemInvalid not in suppColorSystems
— uhAnaVdec_ColorSystemUnknown not in suppColorSystems
— uhAnaVdec_ColorFreqInvalid not in suppColorFrequencies
— uhAnaVdec_ColorFreqUnknown not in suppColorFrequencies

8.1.2.3.3 Instantiation

8.1.2.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 501

Type	Object	Multiplicity
AnalogVideoDecoder	avdec	1

8.1.2.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— avdec.videoPresenceStatus == uhAnaVdec_ValStatInvalid
— avdec.videoPresence == determined at instantiation
— avdec.videoTimingStatus == uhAnaVdec_ValStatInvalid
— avdec.fieldFrequency == determined at instantiation
— avdec.linesPerFrame == determined at instantiation
— avdec.scanType == determined at instantiation
— avdec.videoCoding == determined at instantiation
— avdec.lineFrequency == determined at instantiation
— avdec.protection == determined at instantiation
— avdec.vcrSignalStatus == uhAnaVdec_ValStatInvalid
— avdec.vcrSignalPresence == determined at instantiation
— avdec.pixelsPerLineStatus == uhAnaVdec_ValStatInvalid
— avdec.pixelsPerLine == determined at instantiation
— avdec.hueStatus == uhAnaVdec_ValStatInvalid
— avdec.hue == avd.hueNeutral
— avdec.setColorSystem == uhAnaVdec_ColorSystemInvalid
— avdec.actualColorSystem == uhAnaVdec_ColorSystemInvalid
— avdec.setColorFrequency == uhAnaVdec_ColorFreqInvalid
— avdec.actualColorFrequency == uhAnaVdec_ColorFreqInvalid

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— avdec.combActiveStatus == uhAnaVdec_ValStatInvalid
— avdec.combFilterEnabled == False
— avdec.combFilterActive == False
— avdec.stableOutput == False

8.1.2.3.4 Execution Constraints

The interfaces provided by a decoder may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier “thread-safe”.

8.1.2.4 Roles

8.1.2.4.1 AnalogVideoDecoder

Signature

```
role AnalogVideoDecoder {
    uhAnaVdec_ValStat_t              videoPresenceStatus;
    Bool                             videoPresence;
    uhAnaVdec_ValStat_t              videoTimingStatus;
    Int32                            fieldFrequency;
    Int32                            linesPerFrame;
    uhAnaVdec_ScanType_t             scanType;
    uhTuning_ValStatUnknown          videoCoding;
    uhAnaVdec_LineFreq_t             lineFrequency;
    uhAnaVdec_ValStat_t              vcrSignalStatus;
    Bool                             vcrSignalPresence;
    uhAnaVdec_Protection_t           protection;
    uhAnaVdec_ValStat_t              hueStatus;
    Int32                            hue;
    const Int32                      minHue;
    const Int32                      maxHue;
    const Int32                      hueNeutral;
    Bool                             hueAvailable;
    uhAnaVdec_ColorSystem_t          setColorSystem;
    uhAnaVdec_ColorSystem_t          actualColorSystem;
    uhAnaVdec_ColorFreq_t            setColorFrequency;
    uhAnaVdec_ColorFreq_t            actualColorFrequency;
    uhAnaVdec_ValStat_t              combActiveStatus;
    Bool                             combFilterEnabled;
    Bool                             combFilterActive;
    uhAnaVdec_ValStat_t              pixelsPerLineStatus;
    Int32                            pixelsPerLine;
    const uhAnaVdec_ColorSystemSet_t suppColorSystems;
    const uhAnaVdec_ColorFreqSet_t   suppColorFrequencies;
    Bool                             stableOutput;
}
```

Qualifiers

— root

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
An Analog Video Decoder offers the following functionality:

— Video property observation.

— Color decoding (optional).

— Comb filtering (optional).

— Hue control (optional).

Independent Attributes

Table 502

Name	Description
videoPresenceStatus	Validity state of the video presence property.
videoPresence	Boolean indicating whether a video signal is present on the input.
videoTimingStatus	Validity state of the field frequency and lines per frame properties.
fieldFrequency	The field frequency of the video signal.
linesPerFrame	The number of lines per frame of the video signal.
scanType	The type of scan of the video signal.
videoCoding	The video coding of the video signal.
lineFrequency	The line frequency of the video signal.
vcrSignalStatus	Validity state of the vcr signal presence property.
vcrSignalPresence	Boolean indicating if the input signal is coming from a VCR.
protection	The type of protection used for the video signal.
hueStatus	Validity state of the hue avail property.
hue	The current hue value to be applied.
minHue	The minimum value for hue that can be set (lower limit).
maxHue	The maximum value for hue that can be set (upper limit).
hueAvailable	Boolean indicating if hue can be controlled.
setColorSystem	The color system of the video signal as it was set by the client.
actualColorSystem	The actual color system of the video signal.
setColorFrequency	The color sub-carrier frequency of the video signal as it was set by the client.
actualColorFrequency	The actual color sub-carrier frequency of the video signal.
combActiveStatus	Validity state of the comb filter active property.
combFilterEnabled	Boolean indicating whether the Comb Filter is enabled.
combFilterActive	Boolean indicating whether the Comb Filter is actually active.
pixelsPerLineStatus	Validity state of the pixels per line property.
pixelsPerLine	The number of pixels per line of the video signal.
suppColorSystems	The supported color systems.
suppColorFrequencies	The supported color frequencies.
stableOutput	Indicates whether the analog video decoder is outputting decoded video long enough so that the client can safely unblank the corresponding output at the appropriate logical component(s) downstream (`True` if this is the case, `False` otherwise).

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes

Table 503

Name	Description
`combFilterActive`	Boolean indicating whether the Comb Filter is active.
`hueNeutral`	Neutral hue value (value that implies no signal correction).

Invariants

- if `( combFilterEnabled == False ) combFilterActive == False`
  - `minHue ≤ hue ≤ maxHue`
  - `hueNeutral == (maxHue - minHue) / 2`

Instantiation

An `AnalogVideoDecoder` is instantiated as one object with initial attribute values as defined in subclause 8.1.2.3.3.

Streaming Behavior

The Video Properties are continuously monitored by the component. The type of color system and the color sub-carrier frequency can be set explicitly by the client or can be determined automatically (if supported).

When the Comb Filter is not active the Comb Filter is by-passed and the input stream is not comb filtered. Whether the Comb filter should be activated is determined by the platform. The Comb Filter can only be activated when it has been enabled by the client

Active Behavior

Table 504

Event	Action
A change in video presence has occurred.	Notify `OnVideoPresenceChanged`
A change in the video coding has occurred.	Notify `OnVideoCodingChanged`
A change in the line frequency has occurred.	Notify `OnLineFrequencyChanged`
A change in one or more of the video timing properties occurred.	Notify `OnVideoTimingChanged`
A change in the VCR signal presence has occurred.	Notify `OnVcrSignalPresenceChanged`
A change in the protection of the video signal has occurred.	Notify `OnProtectionChanged`
A change in the active state of the Comb Filter has occurred.	Notify `OnCombFilterChanged`
A change in the Color System has occurred.	Notify `OnColorSystemChanged`
A change in the number of pixels per line has occurred.	Notify `OnPixelsPerLineChanged`
The analog video decoder is outputting decoded video long enough so that the client can safely unblank the corresponding output at the appropriate logical component(s) downstream. (`avdec.stableOutput == False` when this event occurs).	`avdec.stableOutput = True` — notify `OnStableOutput(*, True)`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Event	Action
The analog video decoder is no longer outputting a stable (decoded) video stream (`avdec.stableOutput == True` when this event occurs).	`avdec.stableOutput = False` — notify `OnStableOutput(*, False)`

8.1.2.4.2 AnalogVideoDecoderClient

Signature
```
role AnalogVideoDecoderClient { }
```

Qualifiers
— actor

Description
An `AnalogVideoDecoderClient` role represents a user of the control interfaces offered by the `AnalogVideoDecoder` role. It can optionally provide the notification interfaces to receive notifications of events generated by the `AnalogVideoDecoder` role. No constraints apply to the behavior of the `AnalogVideoDecoderClient` role other than those implied by Figure 63.

8.1.2.5 Interfaces

8.1.2.5.1 uhIAnaVdec

Qualifiers
— None.

Description
This interface can be used to retrieve properties of the analogue video input signal or to subscribe or unsubscribe to notifications.

Interface ID
```
uuid( E23FCCAD-8602-4034-83FD-A3009E455AB2 )
```

8.1.2.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAnaVdecNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAnaVdec_Notification_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIAnaVdecNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAnaVdec_Notification_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.2.5.1.3 GetVideoPresence

Signature

```
uhErrorCode_t GetVideoPresence (
    [out] puhAnaVdec_ValStat_t pStatus,
    [out] pBool                pVideoPresence );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether a video signal is present.

Parameters

Table 505

Name	Description
`pStatus`	Output parameter that indicates the validity of the video presence parameter.
`pVideoPresence`	Video signal presence (`True` indicates present, `False` not present).

Return Values

Standard.

Precondition

`True`

Action

— None.

Postcondition

— `*pStatus == avdec.videoPresenceStatus`

— `*pVideoPresence == avdec.videoPresence`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.5.1.4 GetVideoCoding

Signature
`uhErrorCode_t GetVideoCoding ([out] p`
uhAnaVdec_VideoCoding_t `pVideoCoding );`

Qualifiers
— synchronous

— thread-safe

Description
This function returns the video coding of the video signal.

Parameters

Table 506

Name	Description
`pVideoCoding`	The video coding of the video signal.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
`*pVideoCoding == avdec.videoCoding`

8.1.2.5.1.5 GetLineFrequency

Signature
`uhErrorCode_t GetLineFrequency ([out] puhAnaVdec_LineFreq_t pLineFrequency );`

Qualifiers
— synchronous

— thread-safe

Description
This function returns the line frequency of the video signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 507

Name	Description
pLineFrequency	The line frequency of the video signal.

Return Values
Standard.

Precondition
True

Action
— None.

Postcondition
*pLineFrequency == avdec.lineFrequency

8.1.2.5.1.6 GetVideoTiming

Signature

```
uhErrorCode_t GetVideoTiming (
    [out] puhAnaVdec_ValStat_t  pStatus,
    [out] pInt32                pFieldFrequency,
    [out] pInt32                pLinesPerFrame,
    [out] puhAnaVdec_ScanType_t pScanType
);
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the video timing properties of the video signal.

Parameters

Table 508

Name	Description
pStatus	Output parameter that indicates the validity of the field frequency and lines per frame parameters.
pFieldFrequency	The field frequency in Hz.
pLinesPerFrame	The number of lines per frame.
pScanType	The type of scan.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— None.

Postcondition
— `*pStatus == avdec.videoTimingStatus`
— `*pFieldFrequency == avdec.fieldFrequency`
— `*pLinesPerFrame == avdec.linesPerFrame`
— `*pScanType == avdec.scanType`

8.1.2.5.1.7 GetVcrSignalPresence

Signature
```
uhErrorCode_t GetVcrSignalPresence (
    [out] puhAnaVdec_ValStat_t pStatus,
    [out] pBool                pVcrSignalPresence );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns whether the video input signal is coming from a VCR .

Parameters

Table 509

Name	Description
`pStatus`	Output parameter that indicates the validity of the vcr signal presence parameter.
`pVcrSignalPresence`	If `True` the signal is coming from a VCR.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*pStatus == avdec.vcrSignalStatus`
— `*pVcrSignalPresence == avdec.vcrSignalPresence`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.5.1.8 GetProtection

Signature
```
uhErrorCode_t GetProtection ([out] puhAnaVdec_Protection_t pProtection );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the protection mechanism used for the video signal.

Parameters

Table 510

Name	Description
pProtection	The protection mechanism used.

Return Values
Standard.

Precondition
```
True
```

Action
— None.

Postcondition
```
*pProtection == avdec.protection
```

8.1.2.5.1.9 GetPixelsPerLine

Signature
```
uhErrorCode_t GetPixelsPerLine (
    [out] puhAnaVdec_ValStat_t pStatus,
    [out] pInt32               pPixelsPerLine );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the number of pixels per line.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 511

Name	Description
`pStatus`	Output parameter that indicates the validity of the pixels per line parameter.
`pPixelsPerLine`	The number of pixels per line.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*pStatus == avdec.pixelsPerLineStatus`
— `*pPixelsPerLine == avd.pixelsPerLine`

8.1.2.5.1.10 GetStableOutput

Signature
```
uhErrorCode_t GetStableOutput (
    [out] pBool pStableOutput);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the analog video decoder has currently stable output.

Parameters

Table 512

Name	Description
`pStableOutput`	Output parameter returning whether the analog video decoder has currently stable output.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pStableOutput == avdec.stableOutput
```

8.1.2.5.2 uhIAnaVdecNtf

Qualifiers
— callback

Description
This interface reports changes in the video properties of the signal.

Interface ID
```
uuid( 9B8FFF6E-E6FD-4543-A567-646881342E70 )
```

8.1.2.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAnaVdec_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.2.5.2.2 OnVideoPresenceChanged

Signature
```
[async] Void OnVideoPresenceChanged (
    [in] UInt32              cookie,
    [in] uhAnaVdec_ValStat_t status,
    [in] Bool                videoPresence );
```

Qualifiers
— callback

Description
This funtion gets called when a change in video presence has occured.

Parameters

Table 513

Name	Description
cookie	The value that was passed by the client at subscription time.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
status	Parameter that indicates the validity of the video presence parameter.
videoPresence	The new video presence.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

Postcondition
True

8.1.2.5.2.3 OnVideoCodingChanged

Signature
```
[async] Void OnVideoCodingChanged (
    [in] UInt32 cookie,
    [in]
uhAnaVdec_VideoCoding_t videoCoding );
```

Qualifiers
— callback

Description
This function get called when a change in the video coding has occured.

Parameters

Table 514

Name	Description
cookie	The value that was passed by the client at subscription time.
videoCoding	The new video coding.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

8.1.2.5.2.4 OnLineFrequencyChanged

Signature
```
[async] Void OnLineFrequencyChanged (
    [in] UInt32 cookie,
    [in] uhAnaVdec_LineFreq_t lineFrequency );
```

Qualifiers
— callback

Description
This function get called when a change in the line frequency has occured.

Parameters

Table 515

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`lineFrequency`	The new line frequency.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.2.5.2.5 OnVideoTimingChanged

Signature
```
[async] Void OnVideoTimingChanged (
    [in] UInt32                cookie,
    [in] uhAnaVdec_ValStat_t   status,
    [in] Int32                 fieldFrequency,
    [in] Int32                 linesPerFrame,
    [in] uhAnaVdec_ScanType_t  scanType );
```

Qualifiers
— callback

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function gets called when a change in one or more of the timing properties of the video signal has ocurred.

Parameters

Table 516

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	Parameter that indicates the validity of the field frequency and lines per frame parameters.
`fieldFrequency`	The current field frequency (Hz).
`linesPerFrame`	The current number of lines per frame.
`scanType`	The current scan type.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.2.5.2.6 OnVcrSignalPresenceChanged

Signature
```
[async] Void OnVcrSignalPresenceChanged (
    [in] UInt32              cookie,
    [in] uhAnaVdec_ValStat_t status,
    [in] Bool                vcrSignalPresence );
```

Qualifiers
— callback

Description
This function get called when a change in the fact whether the signal is coming from a VCR or not has occured.

Parameters

Table 517

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	Parameter that indicates the validity of the vcr signal presence parameter.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
vcrSignalPresence	The signal is coming from a VCR (True) or not (False).

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

Postcondition
True

8.1.2.5.2.7 OnProtectionChanged

Signature
```
[async] Void OnProtectionChanged (
    [in] UInt32 cookie,
    [in] uhAnaVdec_Protection_t protection );
```

Qualifiers
— callback

Description
This function gets called when a change in the protection mechanism of the video signal has occured.

Parameters

Table 518

Name	Description
cookie	The value that was passed by the client at subscription time.
protection	The new protection mechanism.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

Postcondition
True

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.5.2.8 OnPixelsPerLineChanged

Signature

```
[async] Void OnPixelsPerLineChanged (
    [in] UInt32              cookie,
    [in] uhAnaVdec_ValStat_t status,
    [in] Int32               pixelsPerLine);
```

Qualifiers
None.

Description
This function gets called when a change occurs in the number of pixels per line.

Parameters

Table 519

Name	Description
cookie	The value that was passed by the client at subscription time.
status	Parameter that indicates the validity of the pixel per line parameter.
pixelsPerLine	The new number of pixels per line.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

Postcondition
True

8.1.2.5.2.9 OnStableOutput

Signature

```
[async] Void OnStableOutput (
    [in] UInt32 cookie,
    [in] Bool stableOutput );
```

Qualifiers
None.

Description
This function gets called when there is a change in the stability of output of the analog video decoder.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 520

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`stableOutput`	If `True`: the analog video decoder is outputting decoded video long enough so that the client can safely unblank the corresponding output at the appropriate logical component(s) downstream. If `False`: the analog video decoder is no longer outputting a stable (decoded) video stream.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
True

8.1.2.5.3 uhIAnaVdecHue

Qualifiers
— None.

Description
This interface offers control to set the hue.

Interface ID
`uuid( 55E44962-4E34-4fea-AE9B-B63CCECF8521 )`

8.1.2.5.3.1 GetHueRange

Signature
```
uhErrorCode_t GetHueRange (
    [out] pInt32 pMinHue,
    [out] pInt32 pMaxHue );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the minimum and maximum hue value that can be set.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 521

Name	Description
`pMinHue`	The minimal hue value to set (lower limit).
`pMaxHue`	The maximal hue value to set (upper limit).

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*pMinHue == avdec.minHue`
— `*pMaxHue == avdec.maxHue`

8.1.2.5.3.2 SetHue

Signature
```
uhErrorCode_t SetHue ( [in] Int32 hue );
```

Qualifiers
— synchronous

Description
This function sets the hue value. If `GetHueAvail` returns `False` the value will have no effect until hue control becomes available (when `GetHueAvail` returns `True`).

Parameters

Table 522

Name	Description
`hue`	The new hue value to be set.

Return Values
Standard.

Precondition
`avdec.minHue ≤ hue ≤ avdec.maxHue`

Action
— Modify `avdec.hue`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`avdec.hue == hue`

8.1.2.5.3.3 GetHue

Signature
```
uhErrorCode_t GetHue ( [out] pInt32 pHue );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the hue value as set by the client.

Parameters

Table 523

Name	Description
`pHue`	The hue value as set by the client.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
`*pHue == avdec.hue`

8.1.2.5.3.4 GetHueAvail

Signature
```
uhErrorCode_t GetHueAvail (
    [out] puhAnaVdec_ValStat_t pStatus,
    [out] pBool                pHueAvail );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns whether hue control is available for the current video signal.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 524

Name	Description
pStatus	Output parameter that indicates the validity of the hue available parameter.
pHueAvail	Indicates whether hue control is available.

Return Values
Standard.

Precondition
True

Action
— None.

Postcondition
— *pStatus == avdec.hueStatus
— *pHueAvail == avdec.hueAvailable

8.1.2.5.4 uhIAnaVdecCombFilter

Qualifiers
None.

Description
This interface provides functionality to control the Comb Filter, retrieve its current state and to subscribe and unsubscribe to notifications.

Interface ID
uuid(CD19C0F9-C5D4-46e9-A2CD-2FEEF9FAF4DE)

8.1.2.5.4.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAnaVdecCombFilterNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAnaVdec_CombFilterNotifications_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.5.4.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAnaVdecCombFilterNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAnaVdec_CombFilterNotifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.2.5.4.3 SetEnable

Signature
```
uhErrorCode_t SetEnable ( [in] Bool enable );
```

Qualifiers
— synchronous

Description
Enables or disables the Comb Filter. If the Comb Filter is enabled it will only be activate if it can be used for the current input signal.

Parameters

Table 525

Name	Description
`enable`	The new enable value (`True` will enable the Comb Filter).

Return Values
Standard.

Precondition
`True`

Action
— Modify `avdec.combFilterEnabled`

Postcondition
`avdec.combFilterEnabled == enable`

8.1.2.5.4.4 GetEnabled

Signature
```
uhErrorCode_t GetEnabled ( [out] pBool pEnabled );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Returns the enable setting of the Comb Filter as set by the client. If the Comb Filter is enabled it will only be activate if it can be used for the current input signal.

Parameters

Table 526

Name	Description
`pEnabled`	Output parameter indicating whether the Comb Filter is enabled.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pEnabled == avdec.combFilterEnabled`

8.1.2.5.4.5 GetActive

Signature
```
uhErrorCode_t GetActive (
    [out] puhAnaVdec_ValStat_t pStatus,
    [out] pBool                pActive );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the Comb Filter is currently active. The Comb Filter can only be active when it has been enabled. An enabled Comb filter will be inactive when the input signal is not to be Comb filtered (determined by the platform) and is by-passed.

Parameters

Table 527

Name	Description
`pStatus`	Output parameter that indicates the validity of the active parameter.
`pActive`	Output parameter indicating whether the Comb Filter is active.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pStatus == avdec.combActiveStatus`
— `*pActive == avdec.combFilterActive`

8.1.2.5.5 uhIAnaVdecCombFilterNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events related to the Comb Filter.

Interface ID
`uuid( 632E02F5-1D71-4a68-AD91-9E404DD18631 )`

8.1.2.5.5.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAnaVdec_CombFilterNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.2.5.5.2 OnCombFilterChanged

Signature
```
[async] Void OnCombFilterChanged (
    [in] UInt32              cookie,
    [in] uhAnaVdec_ValStat_t status,
    [in] Bool                active );
```

Qualifiers
— callback

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is gets called when there is a change in the active state of the enabled Comb Filter. If the Com Filter is not enabled this function is not called.

Parameters

Table 528

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	Parameter that indicates the validity of the active parameter.
`active`	New value indicating whether the Comb Filter is active.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.2.5.6 uhIAnaVdecColDec

Qualifiers
None.

Description
This interface provides functions to control the Color Decoder and to subscribe and unscribe to notifications.

Interface ID
uuid(38958059-6D54-4bb7-A40F-6F332C94D46C)

8.1.2.5.6.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIAnaVdecColDecNtf *pNotify,
    [in] UInt32 cookie,
    [in] uhAnaVdec_ColDecNotifications_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.5.6.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAnaVdecColDecNtf *pNotify,
    [in] UInt32 cookie,
    [in] uhAnaVdec_ColDecNotifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.2.5.6.3 GetSuppColorSystems

Signature
```
uhErrorCode_t GetSuppColorSystems (
    [out] puhAnaVdec_ColorSystemSet_t pColorSystems );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the supported color systems.

Parameters

Table 529

Name	Description
pColorSystems	The supported color systems.

Return Values
Standard

Precondition
True

Action
— None

Postcondition
*pColorSystems == avdec.suppColorSystems

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.2.5.6.4 GetSuppColorFreqs

Signature

```
uhErrorCode_t GetSuppColorFreqs (
    [out] puhAnaVdec_ColorFreqSet_t pColorFreqs );
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns the supported color systems.

Parameters

Table 530

Name	Description
`pColorFreqs`	The supported color frequencies.

Return Values

Standard.

Precondition

```
True
```

Action

— None

Postcondition

```
*pColorFreqs == avdec.suppColorFrequencies
```

8.1.2.5.6.5 GetColorSystemSupported

Signature

```
uhErrorCode_t GetColorSystemSupported (
    [in] uhAnaVdec_ColorSystem_t colorSystem,
    [in] uhAnaVdec_ColorFreq_t colorFrequency,
    [out] pBool pSupported );
```

Qualifiers

— synchronous

— thread-safe

Description

This function checks whether the combination of the color sub-carrier frequency and the color system is supported by the platform. If the color system is SECAM, the color sub-carrier frequency should be the OR-ed value of both color sub-carrier frequency values.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 531

Name	Description
`colorSystem`	The color system.
`colorFrequency`	The color sub-carrier frequency.
`pSupported`	Boolean indicating if the `colorSystem and colorFrequency` combination is supported.

Return Values
Standard.

Precondition
— `colorSystem in avdec.suppColorSystems`
— `colorFrequency in avdec.suppColorFrequencies`

Action
None.

Postcondition
— `if (`the combination of `colorSystem` and `colorFrequency` is supported`)`
  — `*pSupported == True`
  else
  — `*pSupported == False`

8.1.2.5.6.6 SetColorSystem

Signature
```
uhErrorCode_t SetColorSystem (
    [in] uhAnaVdec_ColorSystem_t colorSystem,
    [in] uhAnaVdec_ColorFreq_t colorFrequency );
```

Qualifiers
— synchronous

Description
This function sets the color system and the color sub-carrier frequency. If the `colorSystem`, `colorFrequency` or both are set to auto the value of the parameter set to auto is automatically determined and the result notified. If the color system is SECAM the color sub-carrier frequency should be the OR-ed value of both color sub-carrier frequencies.

Parameters

Table 532

Name	Description
`colorSystem`	The new value of the color system to be used.
`colorFrequency`	The new value of the color sub-carrier frequency to be used.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
— colorSystem in avdec.suppColorSystems
— colorFrequency in avdec.suppColorFrequencies

Action
— Modify avdec.setColorSystem
— Modify avdec.actualColorSystem
— Modify avdec.setColorFrequency
— Modify avdec.actualColorFrequency

Postcondition
— avdec.setColorSystem == colorSystem
— avdec.setColorFrequency == colorFrequency
— if (colorSystem == uhAnaVdec_ColorSystemAuto)
  — avdec.actualColorSystem = uhAnaVdec_ColorSystemUnknown
  else
  — avdec.actualColorSystem = colorSystem
— if (colorFrequency == uhAnaVdec_ColorFreqAuto)
  — avdec.actualColorFrequency = uhAnaVdec_ColorFreqUnknown
  else
  — avdec.actualColorFrequency = colorFrequency

8.1.2.5.6.7 GetColorSystem

Signature
```
uhErrorCode_t GetColorSystem (
    [out] puhAnaVdec_ColorSystem_t pColorSystem,
    [out] puhAnaVdec_ColorFreq_t   pColorFrequency );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the values of the color system and color sub-carrier frequency as set by the client.

Parameters

Table 533

Name	Description
pColorSystem	Output parameter returning the color system as set by the client.
pColorFrequency	Output parameter returning the color sub-carrier frequency as set by the client.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
— `*pColorSystem == avdec.setColorSystem`
— `*pColorFrequency == avdec.setColorFrequency`

8.1.2.5.6.8 GetActualColorSystem

Signature
```
uhErrorCode_t GetActualColorSystem (
    [out] puhAnaVdec_ColorSystem_t pColorSystem,
    [out] puhAnaVdec_ColorFreq_t   pColorFrequency );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the actual color system and color sub-carrier frequency. If the `colorSystem`, `colorFrequency` or both are set to auto the determined value of the parameter set to auto is returned. If determination is still busy or in a transient state `uhAnaVdec_ColorSystemUnknown` and/or `uhAnaVdec_ColorFreqUnknown` is returned. If a value could not be determined `uhAnaVdec_ColorSystemInvalid` and/or `uhAnaVdec_ColorFreqInvalid` is returned. If in the case of the SECAM color system a color sub-carrier frequency value is returned, that frequency is the OR-ed value of both SECAM color sub-carrier frequencies.

Parameters

Table 534

Name	Description
`pColorSystem`	Output parameter returning the actual color system.
`pColorFrequency`	Output parameter returning the actual color sub-carrier frequency.

Return Values
Standard.

Precondition
`True`

Action
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— *pColorSystem == avdec.actualColorSystem
— *pColorFrequency == avdec.actualColorFrequency

8.1.2.5.7 uhIAnaVdecColDecNtf

Qualifiers
— callback

Description
This interface should be provided by each AVDC that wants to subscribe to notifications of events that occur in the Color Decoder. `uhIAnaVdecColDecNtf` is subscribed to via the `Subscribe` function of `uhIAnaVdecColDec`.

Interface ID
`uuid( 647D0CB7-B46A-4a9d-B74B-C3481920F3E9 )`

8.1.2.5.7.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAnaVdec_ColDecNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.2.5.7.2 OnColorSystemChanged

Signature
```
[async] Void OnColorSystemChanged (
    [in] UInt32 cookie,
    [in] uhAnaVdec_ColorSystem_t colorSystem,
    [in] uhAnaVdec_ColorFreq_t   colorFrequency);
```

Qualifiers
— callback

Description
This function gets called when a change in the color system has occurred.

Parameters

Table 535

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`colorSystem`	The current color system.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
colorFrequency	The current color sub-carrier frequency.

Return Values
None.

Precondition
True

Action
Any callback-compliant action.

Postcondition
True

8.1.3 UhlAtscDec

See 7.1.2.

8.1.4 UhlAvIn

8.1.4.1 Concepts

8.1.4.1.1 Introduction

Base band analogue audio and video inputs are often clustered together. Typically, the input signals of a cluster originate from a single source and give access the same signal at different quality levels (i.e. using different types of video coding). In most cases input pins of the socket are used to indicate the available quality levels next to other input signal properties like for instance the aspect ratio of the input signal. Which input signal properties can be detected depend on the actual AV input cluster and hardware configuration. For example: sometimes a S-VHS (YC) socket is part of an AV input cluster and can have the possibility to detect the precence of a plug inserted in the socket instead of the actual presence of a signal indicated by the input source.

An example of a typical AV input cluster is a SCART socket consisting of Y/CVBS and RGB input in combination with one base band stereo input. Several pins (pin 8, pin 16, pin 15 and 20) might be used on a SCART socket to indicate signal properties. Some properties are indicated using the same pin by means of multiple voltage levels of which some can be manufacturer and/or vendor specific. Using these SCART pins it is possible to indicate the following input signal properties:

— Available video coding types of the input signal (i.e. CVBS, RGB, YC)

— Current aspect ratio of the input signal (i.e. 4:3, 16:9, letterbox)

— Input signal presence.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.4.2 Types & Constants

8.1.4.2.1 Public Types & Constants

8.1.4.2.1.1 Error Codes

None.

8.1.4.2.1.2 uhAvIn_Presence_t

Signature

```
typedef enum _uhAvIn_Presence_t {
    uhAvIn_PresenceInvalid    = -2,
    uhAvIn_PresenceUnknown    = -1,
    uhAvIn_PresenceFalse      =  0,
    uhAvIn_PresenceTrue       =  1
} uhAvIn_Presence_t, *puhAvIn_Presence_t;
```

Qualifiers
None.

Description
The detected signal presence on the analog AV input cluster.

Values

Table 536

Name	Description
uhAvIn_PresenceInvalid	The signal presence is temporarily not available.
uhAvIn_PresenceUnknown	The signal presence is available, yet could not be determined.
uhAvIn_PresenceFalse	There is no signal present.
uhAvIn_PresenceTrue	There is a signal present.

8.1.4.2.1.3 uhAvIn_AspectRatio_t

Signature

```
typedef enum _uhAvIn_AspectRatio_t {
    uhAnaAvDem_TvSystemInvalid      = -2,
    uhAnaVdec_VideoCodingInvalid    = -1,
    uhAnaVdec_VideoCodingCVBS       =  0,
    uhAnaVdec_VideoCodingYC         =  1,
    uhAvIn_AspectRatioLetterBox     =  2
} uhAvIn_AspectRatio_t, *puhAvIn_AspectRatio_t;
```

Qualifiers
None.

Description
The detected aspect ratio of the input signal on the analog AV input cluster.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 537

Name	Description
uhAvIn_AspectRatioInvalid	The aspect ratio is temporarily not available.
uhAvIn_AspectRatioUnknown	The aspect ratio is available, yet could not be determined.
uhAvIn_AspectRatio4by3	The signal has a 4 by 3 aspect ratio.
uhAvIn_AspectRatio16by9	The signal has a 16 by 9 (widescreen) aspect ratio.
uhAvIn_AspectRatioLetterBox	The signal has a letterboxed aspect ratio. (16 by 9 content within a 4 by 3 aspect ratio).

8.1.4.2.1.4 uhAvIn_VideoCoding_t

Signature

```
typedef enum _uhAvIn_VideoCoding_t {
    uhAvIn_VideoCodingInvalid= 0x80000000,
    uhAvIn_VideoCodingUnknown = 0x40000000,
    uhAvIn_VideoCodingCvbs    = 0x00000001,
    uhAvIn_VideoCodingYc      = 0x00000002,
    uhAvIn_VideoCodingYuv     = 0x00000004,
    uhAvIn_VideoCodingYPbPr   = 0x00000008,
    uhAvIn_VideoCodingRgb     = 0x00000010
} uhAvIn_VideoCoding_t, *puhAvIn_VideoCoding_t;
```

Qualifiers

— enum-element

Description

The detected video coding of the input signal on the analog AV input cluster.

Values

Table 538

Name	Description
uhAvIn_VideoCodingInvalid	The video coding is temporarily not available.
uhAvIn_VideoCodingUnknown	The video coding is available, yet could not be determined.
uhAvIn_VideoCodingCvbs	CVBS
uhAvIn_VideoCodingYc	Y/C
uhAvIn_VideoCodingYuv	YUV
uhAvIn_VideoCodingYPbPr	YP_BP_R
uhAvIn_VideoCodingRgb	RGB

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.4.2.1.5 uhAvIn_StatusNotification_t

Signature

```
typedef enum _uhAvIn_StatusNotification_t {
    uhAvIn_StatusSubscriptionChanged       = 0x00000001,
    uhAvIn_PresenceChanged                 = 0x00000002,
    uhAvIn_AspectRatioChanged              = 0x00000004,
    uhAvIn_VideoCodingChanged              = 0x00000008
} uhAvIn_StatusNotification_t, *puhAvIn_StatusNotification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIAvInNtf`.

Values

Table 539

Name	Description
`uhAvIn_StatusSubscriptionChanged`	Represents `OnSubscriptionChanged`
`uhAvIn_PresenceChanged`	Represents `OnPresenceChanged`
`uhAvIn_AspectRatioChanged`	Represents `OnAspectRatioChanged`
`uhAvIn_VideoCodingChanged`	Represents `OnVideoCodingChanged`

8.1.4.2.1.6 uhAvIn_StatusNotificationSet_t

Signature

```
typedef Int32 uhAvIn_StatusNotificationSet_t, *puhAvIn_StatusNotificationSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type

uhAvIn_StatusNotification_t. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.4.2.1.7 uhAvIn_StatusAllNotifications

Signature

```
const uhAvIn_StatusNotificationSet_t uhAvIn_StatusAllNotifications = 0x0000000F;
```

Qualifiers
None.

Description
Defines the set of all values of type

uhAvIn_StatusNotification_t. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.1.4.2.2 Model Types & Constants

None.

8.1.4.3 Logical Component

8.1.4.3.1 Interface-Role Model

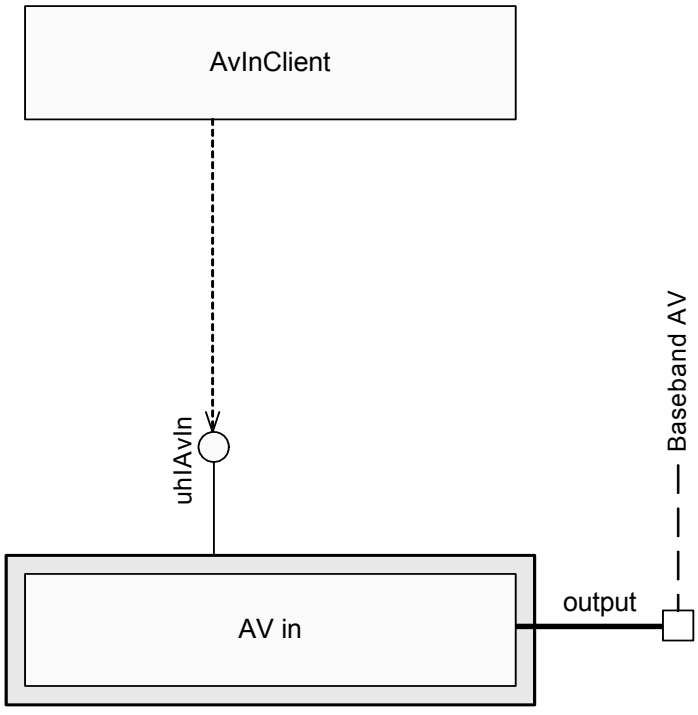


Figure 64 — Interface Role Model

8.1.4.3.2 Diversity

8.1.4.3.2.1 Provided Interfaces

The following table lists the required and provided interfaces:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 540

Role	Interface	Presence
AvIn	uhIAvIn	Mandatory
AvInClient	uhIAvInNtf	optional

8.1.4.3.2.2 Configurable Items

None.

8.1.4.3.2.3 Constraints

None.

8.1.4.3.3 Instantiation

8.1.4.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 541

Type	Object	Multiplicity
AvIn	avin	1

8.1.4.3.3.2 Initial State

```
avin.Presence == uhAvIn_PresenceInvalid

avin.aspectRatio == uhAvIn_AspectRatioInvalid

avin.videoCoding == uhAvIn_VideoCodingInvalid
```

8.1.4.3.4 Execution Constraints

At instantiation of the logical component no notifications are generated to report the initial state. These values must be obtained via the Get functions of the interface(s).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.4.4 Roles

8.1.4.4.1 AvIn

Signature

```
role AvIn {
    uhAvIn_Presence_t    Presence;
    uhAvIn_AspectRatio_t aspectRatio;
    uhAvIn_VideoCoding_t videoCoding;
}
```

Qualifiers

— root

Description

An `AvIn` reports the properties of input signal of the analog AV input cluster.

Dependent Attributes

Table 542

Name	Description
`Presence`	The presence of the input signal.
`aspectRatio`	The aspect ratio of the input signal.
`videoCoding`	The video coding of the input signal.

Invariants

— if `( Presence == uhAvIn_PresenceFalse )`
  — `aspectRatio == uhAvIn_AspectRatioUnknown`
  — `videoCoding == uhAvIn_VideoCodingUnknown`

Instantiation

An `AvIn` is always instantiated as part of an Analog AV Input logical component.

Streaming Behavior

The `AvIn` monitors the properties of the input signal of the analog AV input cluster. Changes in the properties are reported via `uhIAvInNtf`. Current properties can be obtained via `uhIAvIn`.

Active Behavior

Table 543

Event	Action
The presence of the input signal changed.	Modify `avin.Presence`
	Notify `OnPresenceChanged`
The aspect ratio of the input signal changed.	Modify `avin.aspectRatio`
	Notify `OnAspectRatioChanged`
The video coding of the input signal changed.	Modify `avin.videoCoding`
	Notify `OnVideoCodingChanged`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.4.4.2 AvInClient

Signature
role
AvInClient {}

Qualifiers
— actor

Description
— A

— AvInClient role represents a user of the control interfaces offered by the AvIn role. It can optionally provide the notification interfaces to receive notifications of events generated by the AvIn role. No constraints apply to the behavior of the

— AvInClient role other than those implied by Figure 64.

—

8.1.4.5 Interfaces

8.1.4.5.1 uhIAvIn

Qualifiers
— None.

Description
The uhIAvIn interface can be used to obtain the current input signal properties of the analog AV cluster and to subscribe/unsubsrcibe for notifications with respect to property changes.

Interface ID
uuid(41E6F09A-6142-4671-8E47-CEB2DFFD5EE4)

8.1.4.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIAvInNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAvIn_StatusNotificationSet_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.4.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIAvInNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhAvIn_StatusNotificationSet_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.4.5.1.3 GetPresence

Signature
```
uhErrorCode_t GetPresence( [out] puhAvIn_Presence_t pPresence);
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the current presence of the input signal on the analog AV cluster.

Parameters

Table 544

Name	Description
`pPresence`	Output parameter returning the current presence of the input signal.

Precondition
`True`

Action
— None

Postcondition
`*pPresence == avin.Presence`

8.1.4.5.1.4 GetAspectRatio

Signature
```
uhErrorCode_t GetAspectRatio ( [out] puhAvIn_AspectRatio_t pAspectRatio );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the current aspect ratio of the input signal on the analog AV cluster.

Parameters

Table 545

Name	Description
`pAspectRatio`	Output parameter returning the current aspect ratio of the input signal.

Precondition
`True`

Action
— None

Postcondition
`*pAspectRatio == avin.aspectRatio`

8.1.4.5.1.5 GetVideoCoding

Signature
```
uhErrorCode_t GetVideoCoding ( [out] puhAvIn_VideoCoding_t pVideoCoding );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current video coding formats of the input signal on the analog AV cluster.

Parameters

Table 546

Name	Description
`pVideoCoding`	Output parameter returning the current video coding formats of the input signal.

Precondition
`True`

Action
— None

Postcondition
`*pVideoCoding == avin.videoCoding`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.4.5.2 uhIAvInNtf

Qualifiers
— callback

Description
Notifies changes in the input signal properties of the analog AV input cluster.

Interface ID
`uuid( 831D3BD9-A0AF-471e-A7DA-6405736DF5C1 )`

8.1.4.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhAvIn_StatusNotificationSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.4.5.2.2 OnPresenceChanged

Signature
```
[async] Void OnPresenceChanged(
    [in] UInt32 Cookie,
    [in] uhAvIn_Presence_t Presence);
```

Qualifiers
None.

Description
This function is called when a change in the presence or absence of the input signal has been detected.

Parameters

Table 547

Name	Description
`Cookie`	The value that was passed by the client at subscription time.
`Presence`	The new presence of the input signal.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.4.5.2.3 OnAspectRatioChanged

Signature
```
[async] Void OnAspectRatioChanged (
    [in] UInt32 cookie,
    [in] uhAvIn_AspectRatio_t Aspectratio );
```

Qualifiers
None.

Description
This function is called when a change in aspect ratio of the input signal has been detected.

Parameters

Table 548

Name	Description
`Cookie`	The value that was passed by the client at subscription time.
`Aspectratio`	The new aspect ratio of the input signal.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.4.5.2.4 OnVideoCodingChanged

Signature
```
[async] Void OnVideoCodingChanged(
    [in] UInt32 cookie,
    [in] uhAvIn_VideoCoding_t Videocoding );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is called when a change in video coding of the input signal has been detected.

Parameters

Table 549

Name	Description
`Cookie`	The value that was passed by the client at subscription time.
`Videocoding`	The new video coding formats of the input signal.

Return Values
None.

Precondition
`True`

Action
— Any call back compliant action.

Postcondition
`True`

8.1.5 UhlAvOut

8.1.5.1 Concepts

8.1.5.1.1 Introduction

Base band analogue audio and video outputs are often clustered together. Typically, the output signals of a cluster are directed to a single sink and give access the same source signal at different quality levels (i.e. using different types of video coding). In most cases, output pins of the socket are used to indicate/signal the available quality levels next to other output signal properties like for instance the aspect ratio of the output signal. Which output signal properties can be indicated/signalled depend on the actual AV output cluster and hardware configuration. This interface gives a client (if supported) control over the muting of audio and/or video outputs of the AV output cluster and the signalling of video output signal properties.

An example of a typical AV output cluster is a SCART socket consisting of Y/CVBS and RGB output in combination with one base band stereo output. Several pins (pin 8, pin 16, pin 15 and 20) might be used on a SCART socket to indicate/signal signal properties. Some properties are indicated/signalled using the same pin by means of multiple voltage levels of which some can be manufacturer and/or vendor specific. Using these SCART pins it is possible to indicate/signal the following output signal properties:

— Available video coding types of the output signal (i.e. CVBS, RGB, YC). Since signalling of the video conding type is considered the responsibiltiy of the platform no interface is provided to control the signalling of video coding types.

— Aspect ratio of the output signal (i.e. 4:3, 16:9, letterbox).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.5.2 Types & Constants

8.1.5.2.1 Public Types & Constants

8.1.5.2.1.1 Error Codes

— None.

8.1.5.2.1.2 uhAvOut_AspectRatio_t

Signature

```
typedef enum _uhAvOut_AspectRatio_t {
    uhAvOut_AspectRatioUnknown    = 0,
    uhAvOut_AspectRatio4by3       = 1,
    uhAvOut_AspectRatio16by9      = 2,
    uhAvOut_AspectRatioLetterBox  = 3
} uhAvOut_AspectRatio_t, *puhAvOut_AspectRatio_t;
```

Qualifiers
None.

Description
The video aspect ratio types.

Values

Table 550

Name	Description
uhAvOut_AspectRatioUnknown	An unknown aspect ration (determined by the platform).
uhAvOut_AspectRatio4by3	An aspect ration of 4 by 3.
uhAvOut_AspectRatio16by9	An aspect ration of 16 by 9 (widescreen).
uhAvOut_AspectRatioLetterBox	A letterboxed aspect ratio (16 by 9 content within a 4 by 3 aspect ratio).

8.1.5.2.2 Model Types & Constants

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.5.3 Logical Component

8.1.5.3.1 Interface-Role Model

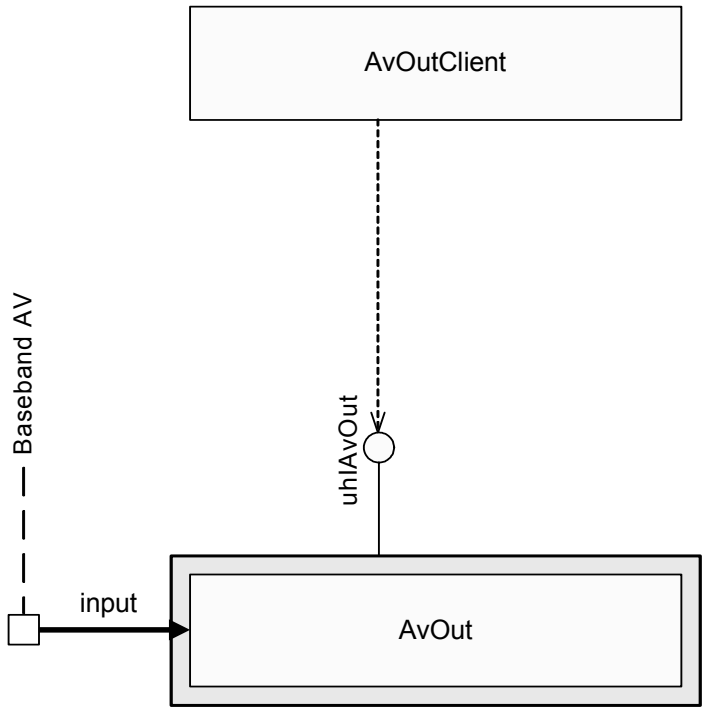


Figure 65 — Interface Role Model

Figure 65 shows the interface role model for the analog AV output.

8.1.5.3.2 Diversity

8.1.5.3.2.1 Provided Interfaces

Table 551

Role	Interface	Presence
AvOut	uhIAvOut	mandatory

8.1.5.3.2.2 Configurable Items

Table 552

Role	Attribute
AvOut	muteVideoSupported
AvOut	muteAudioSupported
AvOut	aspectRatioSupported

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.5.3.2.3 Constraints

— None.

8.1.5.3.3 Instantiation

8.1.5.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 553

Type	Object	Multiplicity
`AvOut`	`avout`	1

8.1.5.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— if `(avout.muteVideoSupported)`
  — `avout.muteVideo == True`
— else
  — `avout.muteVideo == False`

— if `(avout.muteAudioSupported)`
  — `avout.muteAudio == True`
— else
  — `avout.muteAudio == False`
— `avout.aspectRatio == uhAvOut_AspectRatioUnknown`

8.1.5.3.4 Execution Constraints

None.

8.1.5.4 Roles

8.1.5.4.1 AvOut

Signature

```
role AvOut {
    Bool muteVideo;
    const Bool muteVideoSuported;
    Bool muteAudio;
    const Bool muteAudioSupported;
    Bool aspectRatio;
    const Bool aspectRatioSupported;
    }
```

Qualifiers

— Root

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The Analog AV Output controls the baseband audio and/or video output related to a certain analog AV output cluster. If supported, the audio and/or video output can be muted and the signalling of aspect ratio can be controlled.

Independent Attributes

Table 554

Name	Description
`muteVideo`	The mute status of the video of the AV output cluster.
`muteAudio`	The mute status of the audio of the AV output cluster.
`aspectRatio`	The aspect ratio of the video of the AV output cluster.

Invariants
- if `(avout.muteVideo == True)`
  - `avout.muteVideoSupported == True`
- if `( avout.muteAudio == True)`
  - `avout.muteAudioSupported == True`

Instantiation
An `AvOut` is always instantiated as part of an Analog AV Output logical component.

Active Behavior
The `AvOut` has no active behavior.

8.1.5.4.2 AvOutClient

Signature
`role` `AvOutClient {}`

Qualifiers
- actor

Description
An `AvOutClient` role represents a user of the control interface offered by the `AvOut` role. No constraints apply to the behavior of the `AvOutClient` role other than those implied by Figure 65.

8.1.5.5 Interfaces

8.1.5.5.1 uhIAvOut

Qualifiers
- None.

Description
The `uhIAvOut` interface provides control over the muting of the audio and video and control over the signaling of video properties using the analog AV output cluster.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 6155F81B-BA6D-4758-A8B7-43FA46DE16AE )`

8.1.5.5.1.1 VideoMuteSupported

Signature
```
uhErrorCode_t VideoMuteSupported ( [out] pBool pSupported );
```

Qualifiers
— synchronous

— thread-safe

Description
This function indicates if muting of the video output(s) of the output cluster is supported or not.

Parameters

Table 555

Name	Description
`pSupported`	Output parameter indicating if muting is supported (`True`) or not (`False`).

Precondition
`True`

Action
— None.

Postcondition
`*pSupported == avout.muteVideoSupported`

8.1.5.5.1.2 SetVideoMute

Signature
```
uhErrorCode_t SetVideoMute ([in] Bool mute);
```

Qualifiers
— synchronous

Description
This function mutes or unmutes the video output(s) of the output cluster.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 556

Name	Description
`mute`	Input parameter indicating if the video output(s) must be muted (`True`) or unmuted (`False`).

Precondition
```
avout.muteVideoSupported == True
```

Action
— Modify `avout.muteVideo`

Postcondition
```
avout.muteVideo == mute
```

8.1.5.5.1.3 GetVideoMute

Signature
```
uhErrorCode_t GetVideoMute ( [out] pBool pVideoMute );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the mute status of the video output(s) of the output cluster.

Parameters

Table 557

Name	Description
`pVideoMute`	Output parameter returning the mute status of the video output(s) (`True` indicates muted, `False` indicates unmuted).

Precondition
```
True
```

Action
— None.

Postcondition
```
*pVideoMute == avout.muteVideo
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.5.5.1.4 AudioMuteSupported

Signature
```
uhErrorCode_t AudioMuteSupported ( [out] pBool pSupported );
```

Qualifiers
— synchronous

— thread-safe

Description
This function indicates if muting of the audio output(s) of the output cluster is supported or not.

Parameters

Table 558

Name	Description
`pSupported`	Output parameter indicating if muting of the audio output(s) is supported (`True`) or not (`False`).

Precondition
`True`

Action
— None.

Postcondition
```
*pSupported == avout.muteAudioSupported
```

8.1.5.5.1.5 SetAudioMute

Signature
```
uhErrorCode_t SetAudioMute ([in] Bool mute);
```

Qualifiers
— synchronous

Description
This function mutes or unmutes the audio of the output cluster.

Parameters

Table 559

Name	Description
`mute`	Input parameter indicating if the audio output(s) must be muted (`True`) or unmuted (`False`).

Precondition
```
avout.muteAudioSupported == True
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `avout.muteAudio`

Postcondition
`avout.muteAudio == mute`

8.1.5.5.1.6 GetAudioMute

Signature
`uhErrorCode_t GetAudioMute ( [out] pBool pAudioMute );`

Qualifiers
— synchronous

— thread-safe

Description
This function returns the mute status of the audio output(s) of the output cluster.

Parameters

Table 560

Name	Description
`pAudioMute`	Output parameter returning the mute status of the audio output(s) (`True` indicates muted, `False` indicates unmuted).

Precondition
`True`

Action
— None.

Postcondition
`*pAudioMute == avout.audioMute`

8.1.5.5.1.7 AspectRatioSupported

Signature
`uhErrorCode_t AspectRatioSupported ( [out] pBool pSupported );`

Qualifiers
— synchronous

— thread-safe

Description
This function indicates if signaling of the aspect ratio of the video signal of the output cluster is supported or not.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 561

Name	Description
`pSupported`	Output parameter that indicates if signaling of the aspect ratio of the video signal(s) is supported (`True`) or not (`False`).

Precondition
`True`

Action
— None.

Postcondition
`*pSupported == avout.aspectRatioSupported`

8.1.5.5.1.8 SetAspectRatio

Signature
```
uhErrorCode_t SetAspectRatio ([in] uhAvOut_AspectRatio_t aspectRatio);
```

Qualifiers
— synchronous

Description
This function sets the aspect ratio that must be signalled for the video signal of the output cluster.

Parameters

Table 562

Name	Description
`aspectRatio`	Input parameter indicating the aspect ratio of the video signal that must be signalled.

Precondition
`avout.aspectRatioSupported == True`

Action
— Modify `avout.aspectRatio`

Postcondition
`avout.aspectRatio == aspectRatio`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.5.5.1.9 GetAspectRatio

Signature
```
uhErrorCode_t GetAspectRatio ( [out] puhAvOut_AspectRatio_t pAspectRatio );
```

Qualifiers
- synchronous
- thread-safe

Description
This function returns the aspect ratio being signalled for the video signal of the output cluster.

Parameters

Table 563

Name	Description
`pAspectRatio`	Output parameter returning the aspect ratio being signalled.

Precondition
```
True
```

Action
- None.

Postcondition
```
*pAspectRatio == avout.aspectRatio
```

8.1.6 UhlChanDec

8.1.6.1 Concepts

8.1.6.1.1 Introduction

The function of a CHD is to receive an IF, demodulate and decode it, and output the resulting packets.

8.1.6.1.2 Demodulation & Decoding

Channel decoding (also called digital demodulation & decoding) is the process of translating an incoming digital IF to packets that can be demultiplexed and/or descrambled.

A carrier can be modulated by changing it's frequency, phase (PSK), or amplitude (ASK). Frequency and phase modulation are very close modulation techniques, and typically treated as the same type of modulation.

The following figure shows how the signal is modulated.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

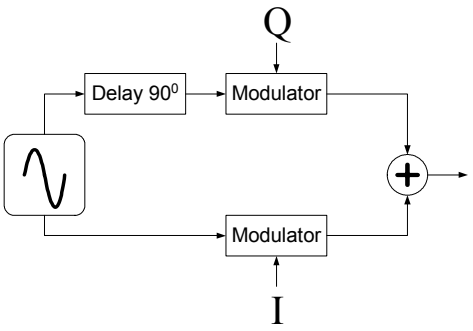


Figure 66 — Typical modulation scheme

A single carrier is split into two paths. The phase of one path is shifted for 90 degrees. Two carriers are amplitude modulated using vector I and vector Q. Possible values for I and Q vectors are normally drawn on two-dimensional graph which is called constellation. If the I axis is the horizontal and Q the vertical one or vice versa determines the IQ mode.

The decoding is channel decoding, which is totally independent from source encoding like MPEG and so on, and consist of analog to digital conversion, filtering, formatting, and forward error correction. The control and level of control for these operations differ between DvbC, DvbT, and DvbS and are described in the following subclauses. The level of control provided by this interface suite is sufficient to implement products based on ATSC and Open Cable standards as well.

During demodulation and decoding, the number of bits that is incorrectly received should be low. An indication for the number of incorrectly received bits is the BER, which denotes the ratio of the number of erroneous bits to the total number of bits transmitted. How low the BER should be is indicated by the BER threshold.

8.1.6.1.3 DVB Cable (DvbC)

DvbC requires little control. In order to do decoding, it is only required to configure the symbol rate correctly, where the symbol rate is the number of symbols transmitted per second. A symbol denotes a number of bits depending on the modulation method that is used. For QPSK a symbol is 2 bits (2^2 == 4, QAM-4 which is the same as QPSK), for QAM-64 it is 6 bits (2^6 = 64). The symbol rate is available in the NIT.

8.1.6.1.4 DVB Terrestrial (DvbT)

Controlling DvbT is more complex. The following information, which is all available in the NIT, has to be provided correctly to the platform:

— **Code Rate**: In the case of terrestrial reception, a high and low priority stream is available. For each of these streams, the inner FEC has to be undone (decoding). For this purpose the code rate has to be configured. The code rate denotes the ratio of the useful bit-rate to the transmitted bit-rate.

— **Hierarchy Mode**: Terrestrial transmission with QAM modulation can be hierarchical and non-hierarchical. In the case of hierarchical transmission, the alpha value has to be configured. This value is the minimum distance separating two constellation points carrying different high-priority bit values divided by the minimum distance separating any two constellation points.

— **Guard Interval**: This is the ratio of the duration of the guard interval (T_g) to the duration of the useful part of an OFDM symbol (T_u). The duration of an OFDM symbol is equal to $T_g + T_u$.

— **Transmission Mode**: This mode determines the number of carriers in an OFDM symbol.

— **Channel Bandwidth**: This is the width of the incoming IF band that has to be demodulated and decoded.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.1.5 DVB Satellite (DvbS)

Controlling DvbS is done by providing the following information, which is available in the NIT, correctly to the platform:

— **Depuncture rate**: This is the scheme needed to undo (decode) the inner FEC of the incoming stream. It denotes the ratio of the useful bit-rate to the transmitted bit-rate.

— **Symbol rate**: This is the number of symbols transmitted per second.

— **Polarization**: This is the polarization of the transmitted signal.

Furthermore, the following LNB related information has to be provided:

— **Transmission band switch mode**: This defines how the transmission band selection of the transmitted signal is effectuated in the LNB that is used.

— **Polarization switch mode**: This defines how the polarization of the transmitted signal is effectuated in the LNB that is used.

— **Transmission band**: This defines the transmission band that is used to select the appropriate local oscillator in the LNB.

— **Power loop through**: If power loop through is enabled, the power to the LNB is supplied from an external receiver.

8.1.6.2 Types & Constants

8.1.6.2.1 Public Types & Constants

8.1.6.2.1.1 Error Codes

None.

8.1.6.2.1.2 uhChanDec_Notification_t

Signature

```
typedef enum _uhChanDec_Notification_t {
    uhChanDec_SubscriptionChanged    = 0x00000001,
    uhChanDec_ConstellationDetected  = 0x00000002,
    uhChanDec_IqModeDetected         = 0x00000004,
    uhChanDec_CodeRateDetected       = 0x00000008,
    uhChanDec_HierarchyModeDetected  = 0x00000010,
    uhChanDec_GuardIntervalDetected  = 0x00000020,
    uhChanDec_TxModeDetected         = 0x00000040,
    uhChanDec_ChanBandwidthDetected  = 0x00000080,
    uhChanDec_DepunctureRateDetected = 0x00000100,
    uhChanDec_PolarizationDetected   = 0x00000200,
    uhChanDec_CarrierPresentChanged  = 0x00000400,
    uhChanDec_BerThresholdCrossed    = 0x00000800
} uhChanDec_Notification_t, *puhChanDec_Notification_t;
```

Qualifiers

— enum-element

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent the notification functions in the `uhIChanDecNtf` interface. They can be used to individually turn on and off notifications.

Values

Table 564

Name	Description
`uhChanDec_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhChanDec_ConstellationDetected`	Represents `OnConstellationDetected`.
`uhChanDec_IqModeDetected`	Represents `OnIqModeDetected`.
`uhChanDec_CodeRateDetected`	Represents `OnCodeRateDetected`.
`uhChanDec_HierarchyModeDetected`	Represents `OnHierarchyModeDetected`.
`uhChanDec_GuardIntervalDetected`	Represents `OnGuardIntervalDetected`.
`uhChanDec_TxModeDetected`	Represents `OnTxModeDetected`.
`uhChanDec_ChanBandwidthDetected`	Represents `OnChanBandwidthDetected`.
`uhChanDec_DepunctureRateDetected`	Represents `OnDepunctureRateDetected`.
`uhChanDec_PolarizationDetected`	Represents `OnPolarizationDetected`.
`uhChanDec_CarrierPresentChanged`	Represents `OnCarrierPresentChanged`.
`uhChanDec_BerThresholdCrossed`	Represents `OnBerThresholdCrossed`.

8.1.6.2.1.3 uhChanDec_Notifications_t

Signature
```
typedef Int32 uhChanDec_Notifications_t, *puhChanDec_Notifications_t;
```

Qualifiers
— enum-set

Description
Defines the set-type for `uhChanDec_Notifications_t`.

8.1.6.2.1.4 uhChanDec_AllNotifications

Signature
```
const uhChanDec_Notifications_t uhChanDec_AllNotifications = 0x00000FFF;
```

Qualifiers
— none

Description
The set of all notifications of `uhChanDec_Notifications_t`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2.1.5 uhChanDec_Constellation_t

Signature

```
typedef enum _uhChanDec_Constellation_t {
    uhChanDec_ConstellationInvalid   = 0x80000000,
    uhChanDec_ConstellationUnknown   = 0x40000000,
    uhChanDec_ConstellationAuto      = 0x00000001,
    uhChanDec_ConstellationBpsk      = 0x00000002,
    uhChanDec_ConstellationQpsk      = 0x00000004,
    uhChanDec_Constellation8psk      = 0x00000008,
    uhChanDec_ConstellationQam16     = 0x00000010,
    uhChanDec_ConstellationQam64     = 0x00000020,
    uhChanDec_ConstellationQam128    = 0x00000040,
    uhChanDec_ConstellationQam256    = 0x00000080,
    uhChanDec_ConstellationQam512    = 0x00000100,
    uhChanDec_ConstellationQam1024   = 0x00000200,
    uhChanDec_Constellation8Vsb      = 0x00000400,
    uhChanDec_Constellation16Vsb     = 0x00000800
} uhChanDec_Constellation_t, *puhChanDec_Constellation_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the constellation.

Values

Table 565

Name	Description
uhChanDec_ConstellationInvalid	The constellation is temporarily not available.
uhChanDec_ConstellationUnknown	The constellation is available, yet could not be determined.
uhChanDec_ConstellationAuto	The constellation is automatically detected.
uhChanDec_ConstellationBpsk	Binary Phase Shift Keying; phase modulation with two states.
uhChanDec_ConstellationQpsk	Phase modulation with four states (equivalent to QAM-4).
uhChanDec_Constellation8psk	8-Phase Shift Keying; phase modulation with eight states.
uhChanDec_ConstellationQam16	QAM with 16 states.
uhChanDec_ConstellationQam64	QAM with 64 states.
uhChanDec_ConstellationQam128	QAM with 128 states.
uhChanDec_ConstellationQam256	QAM with 256 states.
uhChanDec_ConstellationQam512	QAM with 512 states.
uhChanDec_ConstellationQam1024	QAM with 1024 states.
uhChanDec_Constellation8Vsb	VSB with 8 states.
uhChanDec_Constellation16Vsb	VSB with 16 states.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2.1.6 uhChanDec_ConstellationSet_t

Signature
```
typedef Int32 uhChanDec_ConstellationSet_t, *puhChanDec_ConstellationSet_t;
```

Qualifiers
— enum-set

Description
Defines the set-type for `uhChanDec_Constellation_t`.

8.1.6.2.1.7 uhChanDec_IqMode_t

Signature
```
typedef enum _uhChanDec_IqMode_t {
    uhChanDec_IqModeInvalid  = 0x80000000,
    uhChanDec_IqModeUnknown  = 0x40000000,
    uhChanDec_IqModeAuto     = 0x00000001,
    uhChanDec_IqModeNormal   = 0x00000002,
    uhChanDec_IqModeInverse  = 0x00000004
} uhChanDec_IqMode_t, *puhChanDec_IqMode_t;
```

Qualifiers
— enum-element

Description
Values of this type represent the IQ mode.

Values

Table 566

Name	Description
`uhChanDec_IqModeInvalid`	The IQ mode is temporarily not available.
`uhChanDec_IqModeUnknown`	The IQ mode is available, yet could not be determined.
`uhChanDec_IqModeAuto`	The IQ mode is automatically detected.
`uhChanDec_IqModeNormal`	IQ mode in which I is the horizontal axis and Q the vertical.
`uhChanDec_IqModeInverse`	IQ mode in which I is the vertical axis and Q the horizontal.

8.1.6.2.1.8 uhChanDec_IqModeSet_t

Signature
```
typedef Int32 uhChanDec_IqModeSet_t, *puhChanDec_IqModeSet_t;
```

Qualifiers
— enum-set

Description
Defines the set-type for `uhChanDec_IqMode_t`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2.1.9 uhChanDec_CodeRate_t

Signature

```
typedef enum _uhChanDec_CodeRate_t {
    uhChanDec_CodeRateInvalid  = 0x80000000,
    uhChanDec_CodeRateUnknown  = 0x40000000,
    uhChanDec_CodeRateAuto     = 0x00000001,
    uhChanDec_CodeRateNone     = 0x00000002,
    uhChanDec_CodeRate1_2      = 0x00000004,
    uhChanDec_CodeRate2_3      = 0x00000008,
    uhChanDec_CodeRate3_4      = 0x00000010,
    uhChanDec_CodeRate5_6      = 0x00000020,
    uhChanDec_CodeRate6_7      = 0x00000040,
    uhChanDec_CodeRate7_8      = 0x00000080,
    uhChanDec_CodeRate8_9      = 0x00000100
} uhChanDec_CodeRate_t, *puhChanDec_CodeRate_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the inner FEC scheme in case of terrestrial reception.

Values

Table 567

Name	Description
uhChanDec_CodeRateInvalid	The code rate is temporarily not available.
uhChanDec_CodeRateUnknown	The code rate is available, yet could not be determined.
uhChanDec_CodeRateAuto	The code rate is automatically detected.
uhChanDec_CodeRateNone	The code rate cannot be set.
uhChanDec_CodeRate1_2	Code rate 1/2.
uhChanDec_CodeRate2_3	Code rate 2/3.
uhChanDec_CodeRate3_4	Code rate 3/4.
uhChanDec_CodeRate5_6	Code rate 5/6.
uhChanDec_CodeRate6_7	Code rate 6/7.
uhChanDec_CodeRate7_8	Code rate 7/8.
uhChanDec_CodeRate8_9	Code rate 8/9.

8.1.6.2.1.10 uhChanDec_CodeRateSet_t

Signature

```
typedef Int32 uhChanDec_CodeRateSet_t, *puhChanDec_CodeRateSet_t;
```

Qualifiers

— enum-set

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set-type for `uhChanDec_CodeRate_t`.

8.1.6.2.1.11 uhChanDec_HierarchyMode_t

Signature
```
typedef enum _uhChanDec_HierarchyMode_t {
    uhChanDec_HierarchyModeInvalid  = 0x80000000,
    uhChanDec_HierarchyModeUnknown  = 0x40000000,
    uhChanDec_HierarchyModeAuto     = 0x00000001,
    uhChanDec_HierarchyAlpha1       = 0x00000002,
    uhChanDec_HierarchyAlpha2       = 0x00000004,
    uhChanDec_HierarchyAlpha4       = 0x00000008,
    uhChanDec_HierarchyNone         = 0x00000010
} uhChanDec_HierarchyMode_t, *puhChanDec_HierarchyMode_t;
```

Qualifiers
— enum-element

Description
Values of this type represent the hierarchy mode.

Values

Table 568

Name	Description
`uhChanDec_HierarchyModeInvalid`	The hierarchy mode is temporarily not available.
`uhChanDec_HierarchyModeUnknown`	The hierarchy mode is available, yet could not be determined.
`uhChanDec_HierarchyModeAuto`	The hierarchy mode is automatically detected.
`uhChanDec_HierarchyAlpha1`	Hierarchical transmission with alpha is 1.
`uhChanDec_HierarchyAlpha2`	Hierarchical transmission with alpha is 2.
`uhChanDec_HierarchyAlpha4`	Hierarchical transmission with alpha is 4.
`uhChanDec_HierarchyNone`	Non-hierarchical transmission.

8.1.6.2.1.12 uhChanDec_HierarchyModeSet_t

Signature
```
typedef Int32 uhChanDec_HierarchyModeSet_t, *puhChanDec_HierarchyModeSet_t;
```

Qualifiers
— enum-set

Description
Defines the set-type for `uhChanDec_HierarchyModeSet_t`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2.1.13 uhChanDec_GuardInterval_t

Signature

```
typedef enum _uhChanDec_GuardInterval_t {
    uhChanDec_GuardIntervalInvalid  = 0x80000000,
    uhChanDec_GuardIntervalUnknown  = 0x40000000,
    uhChanDec_GuardIntervalAuto     = 0x00000001,
    uhChanDec_GuardInterval1_32     = 0x00000002,
    uhChanDec_GuardInterval1_16     = 0x00000004,
    uhChanDec_GuardInterval1_8      = 0x00000008,
    uhChanDec_GuardInterval1_4      = 0x00000010
} uhChanDec_GuardInterval_t, *puhChanDec_GuardInterval_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the guard interval.

Values

Table 569

Name	Description
uhChanDec_GuardIntervalInvalid	The guard interval is temporarily not available.
uhChanDec_GuardIntervalUnknown	The guard interval is available, yet could not be determined.
uhChanDec_GuardIntervalAuto	The guard interval is automatically detected.
uhChanDec_GuardInterval1_32	Guard interval 1/32.
uhChanDec_GuardInterval1_16	Guard interval 1/16.
uhChanDec_GuardInterval1_8	Guard interval 1/8.
uhChanDec_GuardInterval1_4	Guard interval 1/4.

8.1.6.2.1.14 uhChanDec_GuardIntervalSet_t

Signature

```
typedef Int32 uhChanDec_GuardIntervalSet_t, *puhChanDec_GuardIntervalSet_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhChanDec_GuardInterval_t.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2.1.15 uhChanDec_TxMode_t

Signature

```
typedef enum _uhChanDec_TxMode_t {
    uhChanDec_TxModeInvalid  = 0x80000000,
    uhChanDec_TxModeUnknown  = 0x40000000,
    uhChanDec_TxModeAuto     = 0x00000001,
    uhChanDec_TxMode2k       = 0x00000002,
    uhChanDec_TxMode8k       = 0x00000004
} uhChanDec_TxMode_t, *puhChanDec_TxMode_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the transmission mode.

Values

Table 570

Name	Description
uhChanDec_TxModeInvalid	The transmission mode is temporarily not available.
uhChanDec_TxModeUnknown	The transmission mode is available, yet could not be determined.
uhChanDec_TxModeAuto	The transmission mode is automatically detected.
uhChanDec_TxMode2k	2k transmission mode.
uhChanDec_TxMode8k	8k transmission mode.

8.1.6.2.1.16 uhChanDec_TxModeSet_t

Signature

```
typedef Int32 uhChanDec_TxModeSet_t, *puhChanDec_TxModeSet_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhChanDec_TxMode_t.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2.1.17 uhChanDec_ChanBandwidth_t

Signature

```
typedef enum _uhChanDec_ChanBandwidth_t {
    uhChanDec_ChanBandwidthInvalid  = 0x80000000,
    uhChanDec_ChanBandwidthUnknown  = 0x40000000,
    uhChanDec_ChanBandwidthAuto     = 0x00000001,
    uhChanDec_ChanBandwidth6MHz     = 0x00000002,
    uhChanDec_ChanBandwidth7MHz     = 0x00000004,
    uhChanDec_ChanBandwidth8MHz     = 0x00000008
} uhChanDec_ChanBandwidth_t, *puhChanDec_ChanBandwidth_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the channel bandwidth.

Values

Table 571

Name	Description
uhChanDec_ChanBandwidthInvalid	The channel bandwidth is temporarily not available.
uhChanDec_ChanBandwidthUnknown	The channel bandwidth is available, yet could not be determined.
uhChanDec_ChanBandwidthAuto	The channel bandwidth is automatically detected.
uhChanDec_ChanBandwidth6MHz	6 MHz channel bandwidth.
uhChanDec_ChanBandwidth7MHz	7 MHz channel bandwidth.
uhChanDec_ChanBandwidth8MHz	8 MHz channel bandwidth.

8.1.6.2.1.18 uhChanDec_ChanBandwidthSet_t

Signature

```
typedef Int32 uhChanDec_ChanBandwidthSet_t, *puhChanDec_ChanBandwidthSet_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhChanDec_ChanBandwidth_t.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2.1.19 uhChanDec_DepunctureRate_t

Signature

```
typedef enum _uhChanDec_DepunctureRate_t {
    uhChanDec_DepunctureRateInvalid  = 0x80000000,
    uhChanDec_DepunctureRateUnknown  = 0x40000000,
    uhChanDec_DepunctureRateAuto     = 0x00000001,
    uhChanDec_DepunctureRate1_2      = 0x00000002,
    uhChanDec_DepunctureRate2_3      = 0x00000004,
    uhChanDec_DepunctureRate3_4      = 0x00000008,
    uhChanDec_DepunctureRate5_6      = 0x00000010,
    uhChanDec_DepunctureRate6_7      = 0x00000020,
    uhChanDec_DepunctureRate7_8      = 0x00000040
} uhChanDec_DepunctureRate_t, *puhChanDec_DepunctureRate_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the inner FEC scheme in case of satellite reception.

Values

Table 572

Name	Description
uhChanDec_DepunctureRateInvalid	The depuncture rate is temporarily not available.
uhChanDec_DepunctureRateUnknown	The depuncture rate is available, yet could not be determined.
uhChanDec_DepunctureRateAuto	The depuncture rate is automatically detected.
uhChanDec_DepunctureRate1_2	Depuncture rate 1/2.
uhChanDec_DepunctureRate2_3	Depuncture rate 2/3.
uhChanDec_DepunctureRate3_4	Depuncture rate 3/4.
uhChanDec_DepunctureRate5_6	Depuncture rate 5/6.
uhChanDec_DepunctureRate6_7	Depuncture rate 6/7.
uhChanDec_DepunctureRate7_8	Depuncture rate 7/8.

8.1.6.2.1.20 uhChanDec_DepunctureRateSet_t

Signature

```
typedef Int32 uhChanDec_DepunctureRateSet_t, *puhChanDec_DepunctureRateSet_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhChanDec_DepunctureRate_t.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2.1.21 uhChanDec_PolarizationSwitchMode_t

Signature

```
typedef enum _uhChanDec_PolarizationSwitchMode_t {
    uhChanDec_PolarizationSwitchModeNone    = 0,
    uhChanDec_PolarizationSwitchModeVolts   = 1,
    uhChanDec_PolarizationSwitchModeDiseqc  = 2
} uhChanDec_PolarizationSwitchMode_t, *puhChanDec_PolarizationSwitchMode_t;
```

Qualifiers

— none

Description

Values of this type represent the polarization switch mode.

Values

Table 573

Name	Description
uhChanDec_PolarizationSwitchModeNone	No switch can be done, because only one type of polarization supported.
uhChanDec_PolarizationSwitchModeVolts	Switching is done using either 13V or 18V. uhChanDec_PolarizationLinearVertical and uhChanDec_PolarizationCircularRight coincide with 13V whereas uhChanDec_PolarizationLinearHorizontal and uhChanDec_PolarizationCircularLeft coincide with 18V.
uhChanDec_PolarizationSwitchModeDiseqc	Switching is done according to the DiSEqC standard.

8.1.6.2.1.22 uhChanDec_TxBandSwitchMode_t

Signature

```
typedef enum _uhChanDec_TxBandSwitchMode_t {
    uhChanDec_TxBandSwitchModeNone    = 0,
    uhChanDec_TxBandSwitchModeTone    = 1,
    uhChanDec_TxBandSwitchModeDiseqc  = 2
} uhChanDec_TxBandSwitchMode_t, *puhChanDec_TxBandSwitchMode_t;
```

Qualifiers

None.

Description

Values of this type represent the transmission band switch mode.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 574

Name	Description
uhChanDec_TxBandSwitchModeNone	No switch can be done, because only one transmission band supported.
uhChanDec_TxBandSwitchModeTone	Switching is done by superimposing either 0kHz or 22kHz. uhChanDec_TxBandHigh coincides with 22 KHz and uhChanDec_TxBandLow coincides with 0 KHz.
uhChanDec_TxBandSwitchModeDiseqc	Switching is done according to the DiSEqC standard.

8.1.6.2.1.23 uhChanDec_Polarization_t

Signature

```
typedef enum _uhChanDec_Polarization_t {
    uhChanDec_PolarizationInvalid           = 0x80000000,
    uhChanDec_PolarizationUnknown           = 0x40000000,
    uhChanDec_PolarizationAuto              = 0x00000001,
    uhChanDec_PolarizationLinearHorizontal  = 0x00000002,
    uhChanDec_PolarizationLinearVertical    = 0x00000004,
    uhChanDec_PolarizationCircularLeft      = 0x00000008,
    uhChanDec_PolarizationCircularRight     = 0x00000010
} uhChanDec_Polarization_t, *puhChanDec_Polarization_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the polarization.

Values

Table 575

Name	Description
uhChanDec_PolarizationInvalid	The polarization is temporarily not available.
uhChanDec_PolarizationUnknown	The polarization is available, yet could not be determined.
uhChanDec_PolarizationAuto	The polarization is automatically detected.
uhChanDec_PolarizationLinearHorizontal	The polarization is linear and horizontal.
uhChanDec_PolarizationLinearVertical	The polarization is linear and vertical.
uhChanDec_PolarizationCircularLeft	The polarization is circular and left.
uhChanDec_PolarizationCircularRight	The polarization is circular and right.

8.1.6.2.1.24 uhChanDec_PolarizationSet_t

Signature

```
typedef Int32 uhChanDec_PolarizationSet_t, *puhChanDec_PolarizationSet_t;
```

Qualifiers

— enum-set

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set-type for `uhChanDec_Polarization_t`.

8.1.6.2.1.25 uhChanDec_TxBand_t

Signature
```
typedef enum _uhChanDec_TxBand_t {
    uhChanDec_TxBandHigh = 0x00000001,
    uhChanDec_TxBandLow  = 0x00000002
} uhChanDec_TxBand_t, *puhChanDec_TxBand_t;
```

Qualifiers
— enum-element

Description
Values of this type represent the transmission band.

Values

Table 576

Name	Description
`uhChanDec_TxBandHigh`	The high transmission band is used.
`uhChanDec_TxBandLow`	The low transmission band is used.

8.1.6.2.1.26 uhChanDec_TxBandSet_t

Signature
```
typedef Int32 uhChanDec_TxBandSet_t, *puhChanDec_TxBandSet_t;
```

Qualifiers
— enum-set

Description
Defines the set-type for `uhChanDec_TxBand_t`.

8.1.6.2.1.27 uhChanDec_ValStat_t

Signature
```
typedef enum _uhChanDec_ValStat_t {
    uhChanDec_ValStatInvalid = -2,
    uhChanDec_ValStatUnknown = -1,
    uhChanDec_ValStaKnown    =  1
} uhChanDec_ValStat_t, *puhChanDec_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 577

Name	Description
uhChanDec_ValStatInvalid	The signal strength property is temporarily not available.
uhChanDec_ValStatUnknown	The signal strength property is available, yet could not be determined.
uhChanDec_ValStatKnown	The signal strength property is available and determined.

8.1.6.2.2 Model Types & Constants

None.

8.1.6.3 Logical Component

8.1.6.3.1 Interface-Role Model

Figure 67 depicts the interface-role model of the CHD logical component. It shows the control and notification interfaces and the roles involved with this logical component.

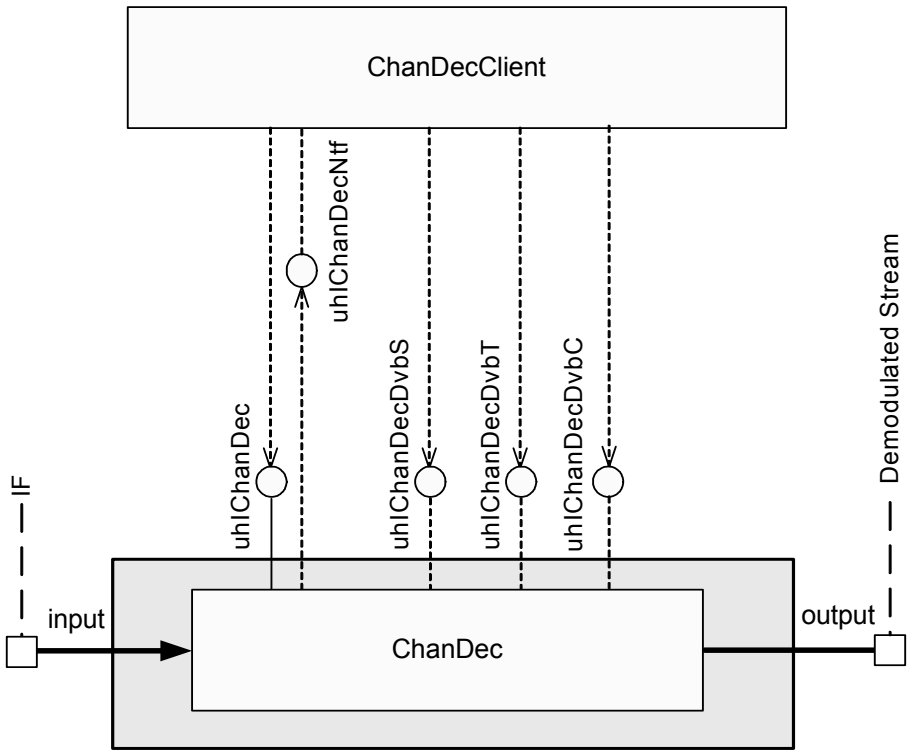


Figure 67 — Interface-Role Model

The grey box in Figure 67 denotes a `CHD` logical component containing the `ChanDec` role providing the mandatory (solid line) and optional (dotted line) control interfaces to the clients.

Clients can optionally provide notifcation interfaces allowing them to receive notifications of events in the `ChanDec`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.3.2 Diversity

8.1.6.3.2.1 Provided Interfaces

Table 578

Role	Interface	Presence
ChanDec	uhIChanDec	mandatory
ChanDec	uhIChanDecDvbC	optional
ChanDec	uhIChanDecDvbT	optional
ChanDec	uhIChanDecDvbS	optional
ChanDec	uhIChanDecNtf	optional

8.1.6.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of the logical component. For a description of the attributes see Roles subclause, and for how they are initialized, see subclause 8.1.6.3.3

Table 579

Role	Attribute
ChanDec	suppConstellations
ChanDec	suppIqModes
ChanDec	minBerThreshold
ChanDec	maxBerThreshold
ChanDec	suppCodeRates
ChanDec	suppHierarchyModes
ChanDec	suppGuardIntervals
ChanDec	suppTxModes
ChanDec	suppChannelBandwidths
ChanDec	suppDepunctureRates
ChanDec	suppNrLocalOscillators
ChanDec	suppPolarizations
ChanDec	suppTxBands
ChanDec	powerLoopThroughSupp

8.1.6.3.2.3 Constraints

— Only one of the uhIChanDecDvbC, uhIChanDecDvbT, and uhIChanDecDvbS interfaces can be present on an instance of the ChanDec role.
— suppConstellations does not contain uhChanDec_ConstellationUnknown
— suppConstellations does not contain uhChanDec_ConstellationInvalid
— suppIqModes does not contain uhChanDec_IqModeUnknown
— suppIqModes does not contain uhChanDec_IqModeInvalid
— minBerThreshold ≤ maxBerThreshold

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— suppCodeRates does not contain uhChanDec_CodeRateUnknown
— suppCodeRates does not contain uhChanDec_CodeRateInvalid
— suppHierarchyModes does not contain uhChanDec_HierarchyModeUnknown
— suppHierarchyModes does not contain uhChanDec_HierarchyModeInvalid
— suppGuardIntervals does not contain uhChanDec_GuardIntervalUnknown
— suppGuardIntervals does not contain uhChanDec_GuardIntervalInvalid
— suppTxModes does not contain uhChanDec_TxModeUnknown
— suppTxModes does not contain uhChanDec_TxModeInvalid
— suppChannelBandwidths does not contain uhChanDec_ChanBandwidthUnknown
— suppChannelBandwidths does not contain uhChanDec_ChanBandwidthInvalid
— suppDepunctureRates does not contain uhChanDec_DepunctureRateUnknown
— suppDepunctureRates does not contain uhChanDec_DepunctureRateInvalid
— suppNrLocalOscillators != 0
— suppPolarizations does not contain uhChanDec_PolarizationUnknown
— suppPolarizations does not contain uhChanDec_PolarizationInvalid

8.1.6.3.3 Instantiation

8.1.6.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 580

Type	Object	Multiplicity
ChanDec	chd	1

8.1.6.3.3.2 Initial State

— chd.setConstellation == uhChanDec_ConstellationUnknown
— chd.actualConstellation == uhChanDec_ConstellationInvalid
— chd.setIqMode == uhChanDec_IqModeUnknown
— chd.actualIqMode == uhChanDec_IqModeInvalid
— chd.setBerThreshold == chd.minBerThreshold
— chd.actualBerStatus == uhChanDec_ValStatInvalid
— chd.actualBer == 0
— chd.carrierPresentStatus == uhChanDec_ValStatInvalid
— chd.symbolRateC == 0
— chd.setHighPrioCodeRate == uhChanDec_CodeRateUnknown
— chd.actualHighPrioCodeRate == uhChanDec_CodeRateInvalid
— chd.setLowPrioCodeRate == uhChanDec_CodeRateUnknown
— chd.actualLowPrioCodeRate == uhChanDec_CodeRateInvalid
— chd.setHierarchyMode == uhChanDec_HierarchyModeUnknown
— chd.actualHierarchyMode == uhChanDec_HierarchyModeInvalid
— chd.setGuardInterval == uhChanDec_GuardIntervalUnknown
— chd.actualGuardInterval == uhChanDec_GuardIntervalInvalid
— chd.setTxMode == uhChanDec_TxModeUnknown
— chd.actualTxMode == uhChanDec_TxModeInvalid
— chd.setChannelBandwidth == uhChanDec_ChanBandwidthUnknown
— chd.actualChannelBandwith == uhChanDec_ChanBandwidthInvalid

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
—— chd.setDepunctureRate == uhChanDec_DepunctureRateUnknown
—— chd.actualDepunctureRate == uhChanDec_DepunctureRateInvalid
—— chd.symbolRateS == 0
—— chd.polarizationSwitchMode == uhChanDec_PolarizationSwitchModeNone
—— chd.txBandSwitchMode == uhChanDec_TxBandSwitchModeNone
—— chd.nrLocalOscillators == chd.suppNrLocalOscillators
—— for i = [0..chd.suppNrLocalOscillators-1]
   —— chd.oscillatorFrequencies[i] == 0
—— chd.setPolarization == uhChanDec_PolarizationUnknown
—— chd.actualPolarization == uhChanDec_PolarizationInvalid
—— chd.setTxBand == undefined
—— chd.powerLoopThroughEnabled == False
——
```

8.1.6.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe.

8.1.6.4 Roles

8.1.6.4.1 ChanDec

Signature

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
role ChanDec {
    const uhChanDec_ConstellationSet_t     suppConstellations;
    uhChanDec_Constellation_t              setConstellation;
    uhChanDec_Constellation_t              actualConstellation;
    const uhChanDec_IqModeSet_t            suppIqModes;
    uhChanDec_IqMode_t                     setIqMode;
    uhChanDec_IqMode_t                     actualIqMode;
    const UInt32                           minBerThreshold;
    const UInt32                           maxBerThreshold;
    UInt32                                 setBerThreshold;
    uhChanDec_ValStat_t                    actualBerStatus;
    UInt32                                 actualBer;
    uhChanDec_ValStat_t                    carrierPresentStatus;
    Bool                                   carrierPresent
    UInt32                                 symbolRateC;
    const uhChanDec_CodeRateSet_t          suppCodeRates;
    uhChanDec_CodeRate_t                   setHighPrioCodeRate;
    uhChanDec_CodeRate_t                   actualHighPrioCodeRate;
    uhChanDec_CodeRate_t                   setLowPrioCodeRate;
    uhChanDec_CodeRate_t                   actualLowPrioCodeRate;
    const uhChanDec_HierarchyModeSet_t     suppHierarchyModes;
    uhChanDec_HierarchyMode_t              setHierarchyMode;
    uhChanDec_HierarchyMode_t              actualHierarchyMode;
    const uhChanDec_GuardIntervalSet_t     suppGuardIntervals;
    uhChanDec_GuardInterval_t              setGuardInterval;
    uhChanDec_GuardInterval_t              actualGuardInterval;
    const uhChanDec_TxModeSet_t            suppTxModes;
    uhChanDec_TxMode_t                     setTxMode;
    uhChanDec_TxMode_t                     actualTxMode;
    const uhChanDec_ChanBandwidthSet_t     suppChannelBandwidths;
    uhChanDec_ChanBandwidth_t              setChannelBandwidth;
    uhChanDec_ChanBandwidth_t              actualChannelBandwith;
    const uhChanDec_DepunctureRateSet_t    suppDepunctureRates;
    uhChanDec_DepunctureRate_t             setDepunctureRate;
    uhChanDec_DepunctureRate_t             actualDepunctureRate;
    UInt32                                 symbolRateS;
    const UInt32                           suppNrLocalOscillators;
    uhChanDec_PolarizationSwitchMode_t     polarizationSwitchMode;
    uhChanDec_TxBandSwitchMode_t           txBandSwitchMode;
    UInt32                                 nrLocalOscillators;
    UInt32                                 oscillatorFrequencies
                                               [nrLocalOscillators];
    const uhChanDec_PolarizationSet_t      suppPolarizations;
    uhChanDec_Polarization_t               setPolarization;
    uhChanDec_Polarization_t               actualPolarization;
    const uhChanDec_TxBandSet_t            suppTxBands;
    uhChanDec_TxBand_t                     setTxBand;
    const Bool                             powerLoopThroughSupp;
    Bool                                   powerLoopThroughEnabled;
}
```

Qualifiers

— actor

Description

A `ChanDec` role controls the demodulation and decoding parameters of the front end.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 581

Name	Description
suppConstellations	The supported constellations.
setConstellation	The constellation as it was set via the SetConstellation function.
actualConstellation	The actual constellation.
suppIqModes	The supported IQ modes.
setIqMode	The IQ mode as it was set via the SetIqMode function.
actualIqMode	The actual IQ mode.
minBerThreshold	The minimum value for BER threshold.
maxBerThreshold	The maximum value for BER threshold.
setBerThreshold	The BER threshold as it was set via the SetBerThreshold function.
actualBerStatus	The validity status of the BER.
actualBer	The actual value of the BER.
carrierPresentStatus	The validity status of the presence of the carrier.
carrierPresent	Presence (True) or absence (False) of the carrier
symbolRateC	The symbol rate for DvbC as it was set via the SetSymbolRate function.
suppCodeRates	The supported code rates.
setHighPrioCodeRate	The code rate for the high priority stream as it was set via the SetCodeRates function.
actualHighPrioCodeRate	The actual code rate for the high priority stream.
setLowPrioCodeRate	The code rate for the low priority stream as it was set via the SetCodeRates function.
actualLowPrioCodeRate	The actual code rate for the low priority stream.
suppHierarchyModes	The supported hierarchy modes.
setHierarchyMode	The hierarchy mode as it was set via the SetHierarchyMode function.
actualHierarchyMode	The actual hierarchy mode.
suppGuardIntervals	The supported guard intervals.
setGuardInterval	The guard interval as it was set via the SetGuardInterval function.
actualGuardInterval	The actual guard interval.
suppTxModes	The supported transmission modes.
setTxMode	The transmission mode as it was set via the SetTxMode function.
actualTxMode	The actual transmission mode.
suppChannelBandwidths	The supported channel bandwidth.
setChannelBandwidth	The channel bandwidth as it was set via the SetChanBandwidth function.
actualChannelBandwith	The actual channel bandwidth.
suppDepunctureRates	The supported depuncture rates.
setDepunctureRate	The depuncture rate as it was set via the SetDepunctureRate function.
actualDepunctureRate	The actual depuncture rate.
symbolRateS	The symbol rate for DvbS as it was set via the SetSymbolRate function.
suppNrLocalOscillators	The supported number of local oscillators.
polarizationSwitchMode	The polarization switch mode as it was set via the ConfigureDish function.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
txBandSwitchMode	The band switch mode as it was set via the ConfigureDish function.
nrLocalOscillators	The number of local oscillators.
oscillatorFrequencies	Array containing the frequencies in Hz of the available local oscillators.
suppPolarizations	The supported polarizations.
setPolarization	The polarization as it was set via the SetPolarization function.
actualPolarization	The actual polarization.
suppTxBands	The supported transmission bands.
setTxBand	The transmission band as it was set via the SetTxBand function.
powerLoopThroughSupp	Indicates if loop through is supported (True) or not (False).
powerLoopThroughEnabled	Indicates if loop through is enabled (True) or disabled (False).

Invariants

- (setConstellation in suppConstellations) || (setConstellation == uhChanDec_ConstellationUnknown)
- (setIqMode in suppIqModes) || (setIqMode == uhChanDec_IqModeUnknown)
- minBerThreshold ≤ setBerThreshold ≤ maxBerThreshold
- (setHighPrioCodeRate in suppCodeRates) || (setHighPrioCodeRate == uhChanDec_CodeRateUnknown)
- (setLowPrioCodeRate in suppCodeRates) || (setLowPrioCodeRate == uhChanDec_CodeRateUnknown)
- (setHierarchyMode in suppHierarchyModes) || (setHierarchyMode == uhChanDec_HierarchyModeUnknown)
- (setGuardInterval in suppGuardIntervals) || (setGuardInterval == uhChanDec_GuardIntervalUnknown)
- (setTxMode in suppTxModes) || (setTxMode == uhChanDec_TxModeUnknown)
- (setChannelBandwidth in suppChannelBandwidths) || (setChannelBandwidth == uhChanDec_ChanBandwidthUnknown)
- (setDepunctureRate in suppDepunctureRates) || (setDepunctureRate == uhChanDec_DepunctureRateUnknown)
- 0 < nrLocalOscillators ≤ suppNrLocalOscillators
- (setPolarization in suppPolarizations) || (setPolarization == uhChanDec_PolarizationUnknown)
- setTxBand in suppTxBands

Instantiation

A ChanDec role is always instantiated as part of a CHD logical component see subclause 8.1.6.3.3.

Streaming Behavior

The ChanDec role demodulates the incoming IF and outputs a signal that can be handled by a demultiplexer.

Active Behavior

The ChanDec has standard notification behavior (see uhIAdrcNtf) except for the following events:

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 582

Event	Action
The platform has determined the constellation after automatic detection was requested.	Modify `actualConstellation` Notify `OnConstellationDetected`
The platform has determined the IQ mode after automatic detection was requested.	Modify `actualIqMode` Notify `OnIqModeDetected`
The platform has determined the code rate after automatic detection was requested.	Modify `actualHighPrioCodeRate` Modify `actualLowPrioCodeRate` Notify `OnCodeRateDetected`
The platform has determined the hierarchy mode after automatic detection was requested.	Modify `actualHierarchyMode` Notify `OnHierarchyModeDetected`
The platform has determined the guard interval after automatic detection was requested.	Modify `actualGuardInterval` Notify `OnGuardIntervalDetected`
The platform has determined the transmission mode after automatic detection was requested.	Modify `actualTxMode` Notify `OnTxModeDetected`
The platform has determined the channel bandwidth after automatic detection was requested.	Modify `actualChannelBandwith` Notify `OnChanBandwidthDetected`
The platform has determined the depuncture rate after automatic detection was requested.	Modify `actualDepunctureRate` Notify `OnDepunctureRateDetected`
The platform has determined the polarization after automatic detection was requested.	Modify `actualPolarization` Notify `OnPolarizationDetected`
The platform has found or lost a carier lock.	Modify `carrierPresent` Notify `OnCarrierPresentChanged`
The BER crossed one of it threshold	Notify `OnBerThresholdCrossed`

8.1.6.4.2 ChanDecClient

Signature

```
role ChanDecClient {}
```

Qualifiers

— actor

Description

A `ChanDecClient` role represents a user of the control interfaces offered by the `ChanDec` role. It can optionally provide the notification interfaces to receive notifications of events generated by the `ChanDec` role. No constraints apply to the behavior of the `ChanDecClient` role other than those implied by Figure 67.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.5 Interfaces

8.1.6.5.1 uhIChanDec

Qualifiers
None.

Description
This interface of a `ChanDec` role provides control of the generic digital demodulation and decoding parameters.

Interface ID
`uuid( 334C6950-A635-4deb-A0DC-622DD61D62BC )`

Remarks

In the following function specifications, the name `chd` will be used to refer to the provider of this interface.

8.1.6.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIChanDecNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhChanDec_Notifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.6.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIChanDecNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhChanDec_Notifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.6.5.1.3 GetSuppConstellations

Signature
```
uhErrorCode_t GetSuppConstellations (
    [out] puhChanDec_ConstellationSet_t pConstellations );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the constellations that are supported.

Parameters

Table 583

Name	Description
`pConstellations`	Output parameter returning the constellations that are supported.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pConstellations == chd.suppConstellations`

8.1.6.5.1.4 SetConstellation

Signature
```
uhErrorCode_t SetConstellation ( [in] uhChanDec_Constellation_t constellation );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the constellation.

Parameters

Table 584

Name	Description
`constellation`	Input parameter indicating the new constellation.

Return Values
Standard.

Precondition
`constellation in chd.suppConstellations`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `chd.setConstellation`

— Modify `chd.actualConstellation`

Postcondition
```
chd.setConstellation == constellation

chd.actualConstellation == constellation
```

8.1.6.5.1.5 GetConstellation

Signature
```
uhErrorCode_t GetConstellation (
    [out] puhChanDec_Constellation_t pConstellation );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the constellation, as it was set via the `SetConstellation` function.

Parameters

Table 585

Name	Description
`pConstellation`	Output parameter returning the constellation that was set.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pConstellation == chd.setConstellation`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.5.1.6 GetActualConstellation

Signature
```
uhErrorCode_t GetActualConstellation (
    [out] puhChanDec_Constellation_t pConstellation );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the actual constellation. If automatic detection is supported and enabled this function will return `uhChanDec_ConstellationAuto` if the detection is busy and `uhChanDec_ConstellationUnknown` if the automatic detection was unsuccessful or has been aborted.

Parameters

Table 586

Name	Description
`pConstellation`	Output parameter returning the actual constellation.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pConstellation == chd.actualConstellation`

8.1.6.5.1.7 GetSuppIqModes

Signature
```
uhErrorCode_t GetSuppIqModes ( [out] puhChanDec_IqModeSet_t pIqModes );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the IQ modes that are supported.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 587

Name	Description
`pIqModes`	Output parameter returning the IQ modes that are supported.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
`*pIqModes == chd.suppIqModes`

8.1.6.5.1.8 SetIqMode

Signature
```
uhErrorCode_t SetIqMode ( [in] uhChanDec_IqMode_t iqMode );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the IQ mode.

Parameters

Table 588

Name	Description
`iqMode`	Input parameter indicating the new IQ mode.

Return Values
Standard.

Precondition
`iqMode in chd.suppIqModes`

Action
— Modify `chd.setIqMode`
— Modify `chd.actualIqMode`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `chd.setIqMode == iqMode`
— `chd.actualIqMode == iqMode`

8.1.6.5.1.9 GetIqMode

Signature
```
uhErrorCode_t GetIqMode ( [out] puhChanDec_IqMode_t pIqMode );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the IQ mode, as it was set via the `SetIqMode` function.

Parameters

Table 589

Name	Description
`pIqMode`	Output parameter returning the IQ mode that was set.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
`*pIqMode == chd.setIqMode`

8.1.6.5.1.10 GetActualIqMode

Signature
```
uhErrorCode_t GetActualIqMode ( [out] puhChanDec_IqMode_t pIqMode );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the actual IQ mode. If automatic detection is supported and enabled this function will return `uhChanDec_IqModeAuto` if the detection is busy and `uhChanDec_IqModeUnknown` if the automatic detection was unsuccessful or has been aborted.

Parameters

Table 590

Name	Description
`pIqMode`	Output parameter returning the actual IQ mode.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
`*pIqMode == chd.actualIqMode`

8.1.6.5.1.11 GetBerThresholdRange

Signature
```
uhErrorCode_t GetBerThresholdRange ( [out] pUInt32 pMinThreshold,
    [out] pUInt32 pMaxThreshold );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the boundaries for the BER threshold in units of 10^{-8}.

Parameters

Table 591

Name	Description
`pMinThreshold`	Output parameter returning the minimum BER threshold in units of 10^{-8}.
`pMaxThreshold`	Output parameter returning the maximum BER threshold in units of 10^{-8}.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— None.

Postcondition
— `*pMinThreshold == chd.minBerThreshold`
— `*pMaxThreshold == chd.maxBerThreshold`

8.1.6.5.1.12 SetBerThreshold

Signature
```
uhErrorCode_t SetBerThreshold ( [in] UInt32 threshold );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the BER threshold in units of 10^{-8}.

Parameters

Table 592

Name	Description
`threshold`	Input parameter indicating the new BER threshold in units of 10^{-8}.

Return Values
Standard.

Precondition
— `chd.minBerThreshold` ≤ *threshold* ≤ `chd.maxBerThreshold`

Action
— Modify `chd.setBerThreshold`

Postcondition
— `chd.setBerThreshold ==` *threshold*

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.5.1.13 GetBerThreshold

Signature
```
uhErrorCode_t GetBerThreshold ( [out] pUInt32 pThreshold );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the BER threshold in units of 10^{-8}, as it was set via the `SetBerThreshold` function.

Parameters

Table 593

Name	Description
`pThreshold`	Output parameter returning the BER threshold in units of 10^{-8}.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
```
*pThreshold == chd.setBerThreshold
```

8.1.6.5.1.14 GetBer

Signature
```
uhErrorCode_t GetBer (
      [out] puhChanDec_ValStat_t pStatus,
      [out] pUInt32 pBer );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the current BER in units of 10^{-8}.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 594

Name	Description
pStatus	Output parameter returning the validity status of the BER.
pBer	Output parameter returning the current BER in units of 10^{-8}.

Return Values
Standard.

Precondition
True

Action
— None.

Postcondition
```
*pStatus == chd.actualBerStatus

*pBer == chd.actualBer
```

8.1.6.5.1.15 GetCarrierPresent

Signature
```
uhErrorCode_t GetVideoPresen (
      [out] puhChanDec_ValStat_t pStatus,
      [out] pBool pPresent );
```

Qualifiers
— synchronous

Description
Returns the presence of a carrier (demodulator has been able to lock onto valid data).

Parameters

Table 595

Name	Description
pStatus	Output parameter returning the validity status of the presence of the carrier.
pPresent	Returns the presence (True) or absence (False) of a carrier.

Precondition
True

Action
— None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— *pStatus == chd.carrierPresentStatus
— *pPresent == chd.carrierPresent

8.1.6.5.2 uhIChanDecNtf

Qualifiers
— callback

Description
This interface can be provided by an instance of the ChanDecClient role that wants to receive notifications of events that occur in the ChanDec. The Subscribe function of the uhIChanDec interface can be used to subscribe to these events.

Interface ID
uuid(4B08494D-2D35-4dda-9FE7-7D9FED4A7A77)

Execution Constraints
— none

8.1.6.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhChanDec_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.6.5.2.2 OnConstellationDetected

Signature
```
[async] Void OnConstellationDetected (
    [in] UInt32 cookie,
    [in] uhChanDec_Constellation_t constellation );
```

Qualifiers
— None

Description
This function notifies subscribed clients that a constellation has been detected.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 596

Name	Description
cookie	The value that was passed by an instance of the ChanDecClient role at subscription time.
constellation	Input parameter indicating detected constellation.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

Postcondition
True

8.1.6.5.2.3 OnIqModeDetected

Signature
```
[async] Void OnIqModeDetected (
    [in] UInt32 cookie,
    [in] uhChanDec_IqMode_t mode );
```

Qualifiers
— None.

Description
This function notifies subscribed clients that the IQ mode has been determined after requesting automatic detection.

Parameters

Table 597

Name	Description
cookie	The value that was passed by an instance of the ChanDecClient role at subscription time.
mode	Input parameter indicating detected IQ mode.

Return Values
None.

Precondition
True

Action
— Any callback compliant action.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

8.1.6.5.2.4 OnCodeRateDetected

Signature
```
[async] Void OnCodeRateDetected (
    [in] UInt32 cookie,
    [in] uhChanDec_CodeRate_t highPrioRate,
    [in] uhChanDec_CodeRate_t lowPrioRate );
```

Qualifiers
— None.

Description
This function notifies subscribed clients that the code rate has been determined after requesting automatic detection. In case only one bit stream is received, this is considered the high priority stream and the code rate for the low priority stream is set to `uhChanDec_CodeRateNone`.

Parameters

Table 598

Name	Description
`cookie`	The value that was passed by an instance of the `ChanDecClient` role at subscription time.
`highPrioRate`	Input parameter indicating detected high priority code rate.
`lowPrioRate`	Input parameter indicating detected low priority code rate.

Return Values
None.

Precondition
`True`

Action
— Any callback compliant action.

Postcondition
`True`

8.1.6.5.2.5 OnHierarchyModeDetected

Signature
```
[async] Void OnHierarchyModeDetected (
    [in] UInt32 cookie,
    [in] uhChanDec_HierarchyMode_t mode );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function notifies subscribed clients that the hierarchy mode has been determined after requesting automatic detection.

Parameters

Table 599

Name	Description
`cookie`	The value that was passed by an instance of the `ChanDecClient` role at subscription time.
`mode`	Input parameter indicating detected hierarchy mode.

Return Values
None.

Precondition
`True`

Action
— Any callback compliant action.

Postcondition
`True`

8.1.6.5.2.6 OnGuardIntervalDetected

Signature
```
[async] Void OnGuardIntervalDetected (
    [in] UInt32 cookie,
    [in] uhChanDec_GuardInterval_t interval );
```

Qualifiers
— None.

Description
This function notifies subscribed clients that the guard interval has been determined after requesting automatic detection.

Parameters

Table 600

Name	Description
`cookie`	The value that was passed by an instance of the `ChanDecClient` role at subscription time.
`interval`	Input parameter indicating the detected guard interval.

Return Values
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback compliant action.

Postcondition
`True`

8.1.6.5.2.7 OnTxModeDetected

Signature
```
[async] Void OnTxModeDetected (
    [in] UInt32 cookie,
    [in] uhChanDec_TxMode_t mode );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the transmission mode has been determined after requesting automatic detection.

Parameters

Table 601

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`mode`	Input parameter indicating the detected transmission mode.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.6.5.2.8 OnChanBandwidthDetected

Signature
```
[async] Void OnChanBandwidthDetected (
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32 cookie,
    [in] uhChanDec_ChanBandwidth_t bandwith );
```

Qualifiers
— None.

Description
This function notifies subscribed clients that the channel bandwidth has been determined after requesting automatic detection.

Parameters

Table 602

Name	Description
`cookie`	The value that was passed by an instance of the `ChanDecClient` role at subscription time.
`bandwidth`	Input parameter indicating detected channel bandwidth.

Return Values
None.

Precondition
`True`

Action
— Any callback compliant action.

Postcondition
`True`

8.1.6.5.2.9 OnDepunctureRateDetected

Signature

```
[async] Void OnDepunctureRateDetected (
    [in] UInt32 cookie,
    [in] uhChanDec_DepunctureRate_t rate );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the depuncture rate has been determined after requesting automatic detection.

Parameters

Table 603

Name	Description
`cookie`	The value that was passed by an instance of the `ChanDecClient` role at subscription time.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
rate	Input parameter indicating detected depuncture rate.

Return Values
None.

Precondition
True

Action
— Any callback compliant action.

Postcondition
True

8.1.6.5.2.10 OnPolarizationDetected

Signature
```
[async] Void OnPolarizationDetected (
    [in] UInt32 cookie,
    [in] uhChanDec_Polarization_t polarization );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the IQ mode has been determined after requesting automatic detection.

Parameters

Table 604

Name	Description
cookie	The value that was passed by an instance of the ChanDecClient role at subscription time.
polarization	Input parameter indicating detected IQ mode.

Return Values
None.

Precondition
True

Action
— Any callback compliant action.

Postcondition
True

-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.5.2.11 OnCarrierPresentChanged

Signature

```
[async] Void OnCarrierPresentChanged (
    [in] UInt32 cookie,
    [in] uhChanDec_ValStat_t status,
    [in] Bool   carrier );
```

Qualifiers
None.

Description
This function notifies subscribed clients that a transition from no carrier to a carrier has been detected on the input. Carrier present means that the demodulator has been able to lock onto valid data on the input.

Parameters

Table 605

Name	Description
`cookie`	The value that was passed by an instance of the `ChanDecClient` role at subscription time.
`status`	The validity status of the notified carrier presence.
`carrier`	The presence (`True`) or absence (`False`) of the carrier.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.6.5.2.12 OnBerThresholdCrossed

Signature

```
[async] Void OnBerThresholdCrossed (
    [in] UInt32 cookie,
    [in] uhChanDec_ValStat_t status,
    [in] UInt32 ber,
    [in] UInt32 berThreshold );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the current BER crossed a BER threshold.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 606

Name	Description
`cookie`	The value that was passed by an instance of the ChanDecClient role at subscription time.
`status`	The validity status of the notified BER.
`ber`	Input parameter returning the BER in units of 10^{-8}.
`berThreshold`	Input parameter returning the BER threshold that was crossed in units of 10^{-8}.

Return Values
None.

Precondition
`True`

Action
— Any callback compliant action.

Postcondition
`True`

8.1.6.5.3 uhIChanDecDvbC

Qualifiers
None.

Description
This interface of a `ChanDec` role provides control of the DVB cable parameters.

Interface ID
`uuid( CB163A49-3747-48d0-AC7B-43B63735118D )`

Remarks

In the following function specifications, the name `chd` will be used to refer to the provider of this interface.

8.1.6.5.3.1 SetSymbolRate

Signature
```
uhErrorCode_t SetSymbolRate ( [in] UInt32 rate );
```

Qualifiers
— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function sets the symbol rate in symbols/second.

Parameters

Table 607

Name	Description
`rate`	Input parameter indicating the new symbol rate in symbols/second.

Return Values
Standard.

Precondition
`True`

Action
— Modify `chd.symbolRateC`

Postcondition
`chd.symbolRateC == rate`

8.1.6.5.3.2 GetSymbolRate

Signature
```
uhErrorCode_t GetSymbolRate ( [out] pUInt32 pRate );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the symbol rate in symbols per second as it was set via the `SetSymbolRate` function.

Parameters

Table 608

Name	Description
`pRate`	Output parameter returning the set symbol rate in symbols per second.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
`*pRate == chd.symbolRateC`

8.1.6.5.4 uhIChanDecDvbT

Qualifiers
None.

Description
This interface of a `ChanDec` role provides control of the DVB terrestrial parameters.

Interface ID
`uuid( BD238F95-5912-4246-84FD-BA480800BB70 )`

Remarks

In the following function specifications the name `chd` will be used to refer to the provider of this interface.

8.1.6.5.4.1 GetSuppCodeRates

Signature
```
uhErrorCode_t GetSuppCodeRates (
    [out] puhChanDec_CodeRateSet_t pRates );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the code rates that are supported.

Parameters

Table 609

Name	Description
`pRates`	Output parameter returning the code rates that are supported.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pRates == chd.suppCodeRates
```

8.1.6.5.4.2 SetCodeRates

Signature
```
uhErrorCode_t SetCodeRates (
    [in] uhChanDec_CodeRate_t highPrioRate,
    [in] uhChanDec_CodeRate_t lowPrioRate );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the code rates for the received high and low priority bit streams. In case only one bit stream is received, this is considered the high priority stream and the code rate for the low priority stream should be set to `uhChanDec_CodeRateNone`.

Parameters

Table 610

Name	Description
`highPrioRate`	Input parameter indicating the code rate for the high priority bit stream.
`lowPrioRate`	Input parameter indicating the code rate for the low priority bit stream.

Return Values
Standard.

Precondition
— `highPrioRate in suppCodeRates`
— `lowPrioRate in suppCodeRates`

Action
— Modify `chd.setHighPrioCodeRate`
— Modify `chd.actualHighPrioCodeRate`
— Modify `chd.setLowPrioCodeRate`
— Modify `chd.actualLowPrioCodeRate`

Postcondition
— `chd.setHighPrioCodeRate == highPrioRate`
— `chd.actualHighPrioCodeRate == highPrioRate`
— `chd.setLowPrioCodeRate == lowPrioRate`
— `chd.actualLowPrioCodeRate == lowPrioRate`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.5.4.3 GetCodeRates

Signature

```
uhErrorCode_t GetCodeRates (
    [out] puhChanDec_CodeRate_t pHighPrioRate,
    [out] puhChanDec_CodeRate_t pLowPrioRate );
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns the code rates of the received high and low priority bit streams, as they were set via the `SetCodeRates` function.

Parameters

Table 611

Name	Description
`pHighPrioRate`	Output parameter returning the code rate of the high priority bit stream.
`pLowPrioRate`	Output parameter returning the code rate of the low priority bit stream.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*pHighPrioRate == chd.setHighPrioCodeRate`

— `*pLowPrioRate == chd.setLowPrioCodeRate`

8.1.6.5.4.4 GetActualCodeRates

Signature

```
uhErrorCode_t GetActualCodeRates (
    [out] puhChanDec_CodeRate_t pHighPrioRate,
    [out] puhChanDec_CodeRate_t pLowPrioRate );
```

Qualifiers

— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the actual code rates for the high and low priority bit stream. If automatic detection is supported and enabled this function will return `uhChanDec_CodeRateAuto` if the detection is busy and `uhChanDec_CodeRateUnknown` if the automatic detection was unsuccessful or has been aborted.

Parameters

Table 612

Name	Description
`pHighPrioRate`	Output parameter returning the actual code rate of the high priority bit stream.
`pLowPrioRate`	Output parameter returning the actual code rate of the low priority bit stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pHighPrioRate == chd.actualHighPrioCodeRate`
— `*pLowPrioRate == chd.actualLowPrioCodeRate`

8.1.6.5.4.5 GetSuppHierarchyModes

Signature
```
uhErrorCode_t GetSuppHierarchyModes (
    [out] puhChanDec_HierarchyModeSet_t pModes );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the hierarchy modes that are supported.

Parameters

Table 613

Name	Description
`pModes`	Output parameter returning the hierarchy modes that are supported.

Return Values
Standard.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
`*pModes == chd.suppHierarchyModes`

8.1.6.5.4.6 SetHierarchyMode

Signature
```
uhErrorCode_t SetHierarchyMode ( [in] uhChanDec_HierarchyMode_t mode );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the hierarchy mode.

Parameters

Table 614

Name	Description
`mode`	Input parameter indicating the new hierarchy mode.

Return Values
Standard.

Precondition
`mode in chd.suppHierarchyModes`

Action
— Modify `chd.setHierarchyMode`
— `Modify chd.actualHierarchyMode`

Postcondition
— `chd.setHierarchyMode == mode`
— `chd.actualHierarchyMode == mode`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.5.4.7 GetHierarchyMode

Signature
```
uhErrorCode_t GetHierarchyMode ( [out] puhChanDec_HierarchyMode_t pMode );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the hierarchy mode, as it was set via the `SetHierarchyMode` function.

Parameters

Table 615

Name	Description
`pMode`	Output parameter returning the hierarchy mode that was set.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
```
*pMode == chd.setHierarchyMode
```

8.1.6.5.4.8 GetActualHierarchyMode

Signature
```
uhErrorCode_t GetActualHierarchyMode ( [out] puhChanDec_HierarchyMode_t pMode );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the actual hierarchy mode. If automatic detection is supported and enabled this function will return `uhChanDec_HierarchyModeAuto` if the detection is busy and `uhChanDec_HierarchyModeUnknown` if the automatic detection was unsuccessful or has been aborted.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 616

Name	Description
pMode	Output parameter returning the actual hierarchy mode.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pMode == chd.actualHierarchyMode

8.1.6.5.4.9 GetSuppGuardIntervals

Signature

```
uhErrorCode_t GetSuppGuardIntervals (
    [out] puhChanDec_GuardIntervalSet_t pIntervals );
```

Qualifiers
- synchronous
- thread-safe

Description
This function returns the guard intervals that are supported.

Parameters

Table 617

Name	Description
pIntervals	Output parameter returning the guard intervals that are supported.

Return Values
Standard.

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pIntervals == chd.suppGuardIntervals`

8.1.6.5.4.10 SetGuardInterval

Signature
```
uhErrorCode_t SetGuardInterval( [in] uhChanDec_GuardInterval_t interval );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the guard interval.

Parameters

Table 618

Name	Description
`interval`	Input parameter indicating the new guard interval.

Return Values
Standard.

Precondition
`interval in chd.suppGuardIntervals`

Action
— Modify `chd.setGuardInterval`
— Modify `chd.actualGuardInterval`

Postcondition
— `chd.setGuardInterval == interval`
— `chd.actualGuardInterval == interval`

8.1.6.5.4.11 GetGuardInterval

Signature
```
uhErrorCode_t GetGuardInterval ( [out] puhChanDec_GuardInterval_t pInterval );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the guard interval, as it was set via the `SetGuardInterval` function.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 619

Name	Description
pInterval	Output parameter returning the guard interval that was set.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pInterval == chd.setGuardInterval

8.1.6.5.4.12 GetActualGuardInterval

Signature

```
uhErrorCode_t GetActualGuardInterval (
    [out] puhChanDec_GuardInterval_t pInterval );
```

Qualifiers
- synchronous
- thread-safe

Description
This function returns the actual guard interval. If automatic detection is supported and enabled this function will return uhChanDec_GuardIntervalAuto if the detection is busy and uhChanDec_GuardIntervalUnknown if the automatic detection was unsuccessful or has been aborted.

Parameters

Table 620

Name	Description
pInterval	Output parameter returning the actual guard interval.

Return Values
Standard.

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pInterval == chd.actualGuardInterval`

8.1.6.5.4.13 GetSuppTxModes

Signature
```
uhErrorCode_t GetSuppTxModes ( [out] puhChanDec_TxModeSet_t pModes );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the transmission modes that are supported.

Parameters

Table 621

Name	Description
`pModes`	Output parameter returning the transmission modes that are supported.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pModes == chd.suppTxModes`

8.1.6.5.4.14 SetTxMode

Signature
```
uhErrorCode_t SetTxMode ( [in] uhChanDec_TxMode_t mode );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the transmission mode.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 622

Name	Description
`mode`	Input parameter indicating the new transmission mode.

Return Values
Standard.

Precondition
— `mode` in `chd.suppTxModes`

Action
— Modify `chd.setTxMode`
— Modify `chd.actualTxMode`

Postcondition
— `chd.setTxMode == mode`
— `chd.actualTxMode == mode`

8.1.6.5.4.15 GetTxMode

Signature
```
uhErrorCode_t GetTxMode ( [out] puhChanDec_TxMode_t pMode );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the transmission mode, as it was set via the `SetTxMode` function.

Parameters

Table 623

Name	Description
`pMode`	Output parameter returning the transmission mode that was set.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pMode == chd.setTxMode`

8.1.6.5.4.16 GetActualTxMode

Signature
```
uhErrorCode_t GetActualTxMode ( [out] puhChanDec_TxMode_t pMode );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the actual transmission mode. If automatic detection is supported and enabled this function will return `uhChanDec_TxModeAuto` if the detection is busy and `uhChanDec_TxModeUnknown` if the automatic detection was unsuccessful or has been aborted.

Parameters

Table 624

Name	Description
`pMode`	Output parameter returning the actual transmission mode.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pMode == chd.actualTxMode`

8.1.6.5.4.17 GetSuppChanBandwidths

Signature
```
uhErrorCode_t GetSuppChanBandwidths (
    [out] puhChanDec_ChanBandwidthSet_t pBandwidths );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the channel bandwidths that are supported.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 625

Name	Description
`pBandwidths`	Output parameter returning the channel bandwidths that are supported.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pBandwidths == chd.suppChannelBandwidths`

8.1.6.5.4.18 SetChanBandwidth

Signature
```
uhErrorCode_t SetChanBandwidth ( [in] uhChanDec_ChanBandwidth_t bandwidth );
```

Qualifiers
- synchronous
- single-threaded

Description
This function sets the channel bandwidth.

Parameters

Table 626

Name	Description
`bandwidth`	Input parameter indicating the new channel bandwidth.

Return Values
Standard.

Precondition
`bandwidth in chd.suppChannelBandwidths`

Action
- Modify `chd.setChannelBandwidth`
- Modify `chd.actualChannelBandwith`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `chd.setChannelBandwidth == bandwidth`
— `chd.actualChannelBandwith == bandwidth`

8.1.6.5.4.19 GetChanBandwidth

Signature
```
uhErrorCode_t GetChanBandwidth ( [out] puhChanDec_ChanBandwidth_t pBandwidth );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the channel bandwidth, as it was set via the `SetChanBandwidth` function.

Parameters

Table 627

Name	Description
`pBandwidth`	Output parameter returning the channel bandwidth that was set.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pBandwidth == chd.setChannelBandwidth`

8.1.6.5.4.20 GetActualChanBandwidth

Signature
```
uhErrorCode_t GetActualChanBandwidth (
    [out] puhChanDec_ChanBandwidth_t pBandwidth );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the actual channel bandwidth. If automatic detection is supported and enabled this function will return `uhChanDec_ChanBandwidthAuto` if the detection is busy and `uhChanDec_ChanBandwidthUnknown` if the automatic detection was unsuccessful or has been aborted.

Parameters

Table 628

Name	Description
`pBandwidth`	Output parameter returning the actual channel bandwidth.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pBandwidth == chd.actualChannelBandwith`

8.1.6.5.5 uhIChanDecDvbS

Qualifiers
None.

Description
This interface of a `ChanDec` role provides functionality to control the DVB satellite parameters.

Interface ID
`uuid( 1CEA64C2-E4A4-4eca-B07B-28B3454C6A00 )`

Remarks
— In the following function specifications the name `chd` will be used to refer to the provider of this interface.

8.1.6.5.5.1 GetSuppDepunctureRates

Signature
```
uhErrorCode_t GetSuppDepunctureRates (
    [out] puhChanDec_DepunctureRateSet_t pRates );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the depuncture rates that are supported.

Parameters

Table 629

Name	Description
`pRates`	Output parameter returning the depuncture rates that are supported.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pRates == chd.suppDepunctureRates`

8.1.6.5.5.2 SetDepunctureRate

Signature
```
uhErrorCode_t SetDepunctureRate ( [in] uhChanDec_DepunctureRate_t rate );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the depuncture rate.

Parameters

Table 630

Name	Description
rate	Input parameter indicating the new depuncture rate.

Return Values
Standard.

Precondition
— `rate in chd.suppDepunctureRates`

Action
— Modify `chd.setDepunctureRate`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Modify `chd.actualDepunctureRate`

Postcondition
— `chd.setDepunctureRate == rate`
— `chd.actualDepunctureRate == rate`

8.1.6.5.5.3 GetDepunctureRate

Signature
```
uhErrorCode_t GetDepunctureRate ( [out] puhChanDec_DepunctureRate_t pRate );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the depuncture rate, as it was set via the `SetDepunctureRate` function.

Parameters

Table 631

Name	Description
`pRate`	Output parameter returning the depuncture rate that was set.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pRate  == chd.setDepunctureRate`

8.1.6.5.5.4 GetActualDepunctureRate

Signature
```
uhErrorCode_t GetActualDepunctureRate (
    [out] puhChanDec_DepunctureRate_t pRate );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the actual depuncture rate. If automatic detection is supported and enabled this function will return `uhChanDec_DepunctureRateAuto` if the detection is busy and `uhChanDec_DepunctureRateUnknown` if the automatic detection was unsuccessful or has been aborted.

Parameters

Table 632

Name	Description
`pRate`	Output parameter returning the actual depuncture rate.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pRate == chd.actualDepunctureRate`

8.1.6.5.5.5 SetSymbolRate

Signature
```
uhErrorCode_t SetSymbolRate ( [in] UInt32 rate );
```

Qualifiers
- synchronous
- single-threaded

Description
This function sets the symbol rate in symbols/second.

Parameters

Table 633

Name	Description
`rate`	Input parameter indicating the new symbol rate in symbols/second.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `chd.symbolRateS`

Postcondition
`chd.symbolRateS == rate`

8.1.6.5.5.6 GetSymbolRate

Signature
`uhErrorCode_t GetSymbolRate ( [out] pUInt32 pRate );`

Qualifiers
— synchronous

— thread-safe

Description
This function returns the symbol rate in symbols per second, as it was set via the `SetSymbolRate` function.

Parameters

Table 634

Name	Description
`pRate`	Output parameter returning the set symbol rate in symbols per second.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pRate == chd.symbolRateS`

8.1.6.5.5.7 GetSuppNrLocalOscillators

Signature
`uhErrorCode_t GetSuppNrLocalOscillators ( [out] pUInt32 pNr );`

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the number of local oscillators that are supported.

Parameters

Table 635

Name	Description
pNr	Output parameter returning the number of local oscillators that are supported.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pNr == chd.suppNrLocalOscillators

8.1.6.5.5.8 ConfigureDish

Signature
```
uhErrorCode_t ConfigureDish (
    [in] uhChanDec_TxBandSwitchMode_t txBandSwitchMode,
    [in] uhChanDec_PolarizationSwitchMode_t polarizationSwitchMode,
    [in] UInt32 nrLocalOscillators,
    [in, size_is(nrLocalOscillators)] const pUInt32 pOscillatorFrequencies );
```

Qualifiers
- synchronous
- single-threaded

Description
This function configures the satellite dish that is currently being used.

Parameters

Table 636

Name	Description
txBandSwitchMode	Input parameter indicating the way transmission band switching is done.
polarizationSwitchMode	Input parameter indicating the way polarization switching is done.
nrLocalOscillators	Input parameter indicating the number of local oscillators that must be configured.
pOscillatorFrequencies	Input parameter indicating the frequency in Hz for each of the local oscillators that has to be configured.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
— `0 < nrLocalOscillators ≤ chd.suppNrLocalOscillators`

Action
— Modify `chd.txBandSwitchMode`
— Modify `chd.polarizationSwitchMode`
— Modify `chd.nrLocalOscillators`
— Modify `chd.oscillatorFrequencies`

Postcondition
— `chd.txBandSwitchMode == txBandSwitchMode`
— `chd.polarizationSwitchMode == polarizationSwitchMode`
— `chd.nrLocalOscillators == nrLocalOscillators`
— for `i = [0..nrLocalOscillators-1]`
  — `chd.oscillatorFrequencies[i] == pOscillatorFrequencies[i]`

8.1.6.5.5.9 GetSuppPolarizations

Signature
```
uhErrorCode_t GetSuppPolarizations (
    [out] puhChanDec_PolarizationSet_t pPolarizations );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the polarizations that are supported.

Parameters

Table 637

Name	Description
`pPolarizations`	Output parameter returning the polarizations that are supported.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pPolarizations == chd.suppPolarizations`

8.1.6.5.5.10 SetPolarization

Signature
```
uhErrorCode_t SetPolarization ( [in] uhChanDec_Polarization_t polarization );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the polarization.

Parameters

Table 638

Name	Description
`polarization`	Input parameter indicating the new polarization.

Return Values
Standard.

Precondition
— `polarization in chd.suppPolarizations`

Action
— Modify `chd.setPolarization`
— Modify `chd.actualPolarization`

Postcondition
— `chd.setPolarization == polarization`
— `chd.actualPolarization == polarization`

8.1.6.5.5.11 GetPolarization

Signature
```
uhErrorCode_t GetPolarization ( [out] puhChanDec_Polarization_t pPolarization );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the polarization, as it was set via the `SetPolarization` function.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 639

Name	Description
pPolarization	Output parameter returning the polarization that was set.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pPolarization == chd.setPolarization

8.1.6.5.5.12 GetActualPolarization

Signature

```
uhErrorCode_t GetActualPolarization (
    [out] puhChanDec_Polarization_t pPolarization );
```

Qualifiers
- synchronous
- thread-safe

Description
This function returns the actual polarization. If automatic detection is supported and enabled this function will return uhChanDec_PolarizationAuto if the detection is busy and uhChanDec_PolarizationUnknown if the automatic detection was unsuccessful or has been aborted.

Parameters

Table 640

Name	Description
pPolarization	Output parameter returning the actual polarization.

Return Values
Standard.

Precondition
True

Action
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`*pPolarization == chd.actualPolarization`

8.1.6.5.5.13 GetSuppTxBands

Signature
```
uhErrorCode_t GetSuppTxBands ( [out] puhChanDec_TxBandSet_t pBands);
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the transmission bands that are supported.

Parameters

Table 641

Name	Description
`pBands`	Output parameter returning the transmission bands that are supported.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pBands == chd.suppTxBands`

8.1.6.5.5.14 SetTxBand

Signature
```
uhErrorCode_t SetTxBand ( [in] uhChanDec_TxBand_t band );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the transmission band to high or low.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 642

Name	Description
band	Input parameter indicating the new transmission band.

Return Values
Standard.

Precondition
band in chd.suppTxBands

Action
— Modify chd.setTxBand

Postcondition
chd.setTxBand == band

8.1.6.5.5.15 GetTxBand

Signature
```
uhErrorCode_t GetTxBand ( [out] puhChanDec_TxBand_t pBand );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the transmission band, as it was set via the SetTxBand function.

Parameters

Table 643

Name	Description
pBand	Output parameter returning the transmission band that was set.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pBand == chd.setTxBand

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.5.5.16 GetPowerLoopThroughSupp

Signature
```
uhErrorCode_t GetPowerLoopThroughSupp ( [out] pBool pSupp );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns whether or not power loop through is supported.

Parameters

Table 644

Name	Description
`pSupp`	Output parameter returning if power loop through is supported (`True`) or not (`False`).

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pSupp == chd.powerLoopThroughSupp`

8.1.6.5.5.17 EnablePowerLoopThrough

Signature
```
uhErrorCode_t EnablePowerLoopThrough ( [in] Bool enable );
```

Qualifiers
— synchronous

— single-threaded

Description
This function enables (`True`) or disables (`False`) loop through.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 645

Name	Description
`enable`	Input parameter indicating if loop through has to be enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
`chd.powerLoopThroughSupp == True`

Action
— Modify `chd.powerLoopThroughEnabled`

Postcondition
`chd.powerLoopThroughEnabled == enable`

8.1.6.5.5.18 GetPowerLoopThroughEnabled

Signature
```
uhErrorCode_t GetPowerLoopThroughEnabled ( [out] pBool pEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
The function returns whether or not power loop through is currently enabled.

Parameters

Table 646

Name	Description
`pEnabled`	Output parameter returning if power loop through is currently enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
```
*pEnabled == chd.powerLoopThroughEnabled
```

8.1.7 UhlHdmiIn

8.1.7.1 Concepts

8.1.7.1.1 Introduction

The input for the HdmiIn is an uncompressed digital input stream, which can contain HDMI InfoFrames. The HdmiIn extracts the InfoFrames from the HDMI stream and passes them on to the client. InfoFrames of several types can be extracted. The variety of InfoFrames that can be extracted depends on the source, which provides the HDMI stream.

The following InfoFrames can be captured:

— Auxiliary Video Information InfoFrame

— Audio InfoFrame

— HDMI Source InfoFrame

— Source Product Description InfoFrame

— Vendor Specific InfoFrame

The HdmiIn can notify the client whenever new InfoFrames of a HDMI stream are captured, see Figure 68.

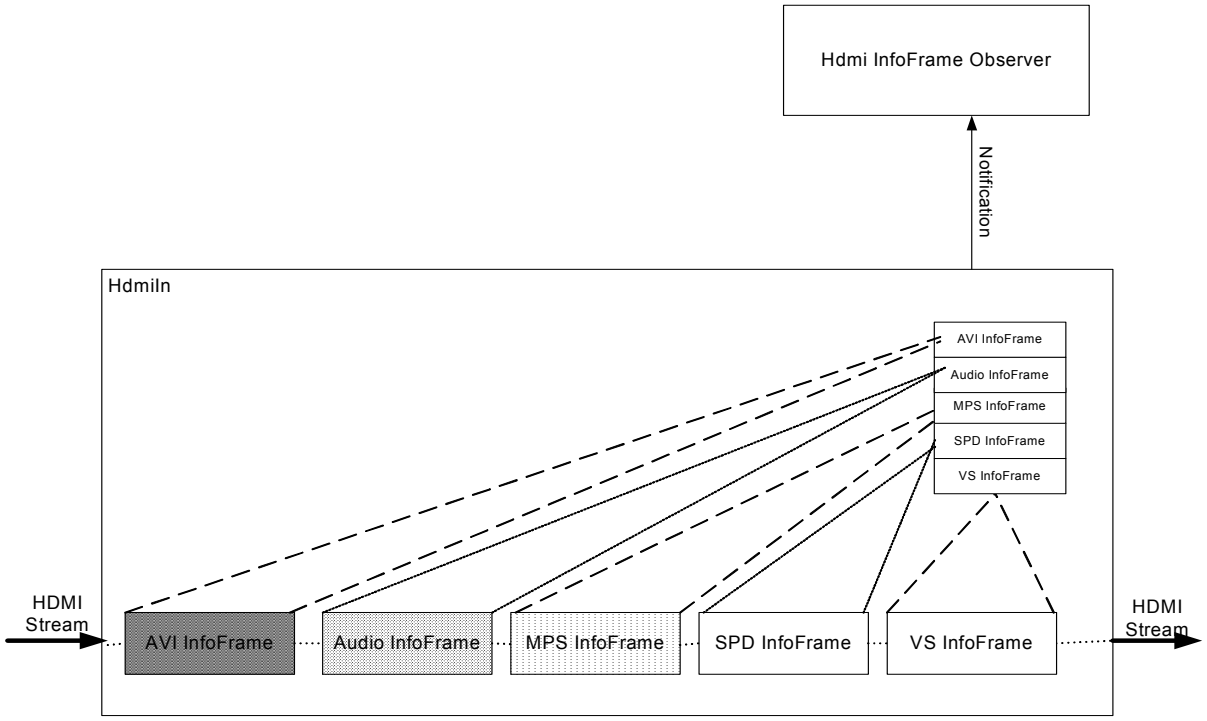


Figure 68 — Behavior of the HdmiIn

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.7.2 Types & Constants

8.1.7.2.1 Public Types & Constants

8.1.7.2.1.1 uhHdmiIn_Notification_t

Signature

```
typedef enum _uhHdmiIn_Notification_t {
    uhHdmiIn_SubscriptionChanged = 0x00000001,
    uhHdmiIn_AviInfoFrame        = 0x00000002,
    uhHdmiIn_AudioInfoFrame      = 0x00000004,
    uhHdmiIn_MSInfoFrame         = 0x00000008,
    uhHdmiIn_SpdInfoFrame        = 0x00000010,
    uhHdmiIn_VsInfoFrame         = 0x00000020
} uhHdmiIn_Notification_t, *puhHdmiIn_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIHdmiInNtf`.

Values

Table 647

Name	Description
`uhHdmiIn_SubscriptionChanged`	Represents `OnSubscriptionChanged`
`uhHdmiIn_AviInfoFrame`	Represents `OnAviInfoFrame`
`uhHdmiIn_AudioInfoFrame`	Represents `OnAudioInfoFrame`
`uhHdmiIn_MSInfoFrame`	Represents `OnMsInfoFrame`
`uhHdmiIn_SpdInfoFrame`	Represents `OnSpdInfoFrame`
`uhHdmiIn_VsInfoFrame`	Represents `OnVsInfoFrame`

8.1.7.2.1.2 uhHdmiIn_Notifications_t

Signature

```
typedef Int32 uhHdmiIn_Notifications_t, *puhHdmiIn_Notifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type `uhHdmiIn_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Constraints
None

8.1.7.2.1.3 uhHdmiIn_AllNotifications

Signature
```
const uhHdmiIn_Notifications_t uhHdmiIn_AllNotifications = 0x00000003F;
```

Qualifiers
None

Description
Defines the set of all values of type `uhHdmiIn_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.1.7.2.1.4 uhHdmiIn_ValStat_t

Signature
```
typedef enum _uhHdmiIn_ValStat_t {
    uhHdmiIn_ValStatInvalid = -2,
    uhHdmiIn_ValStatKnown   =  1
} uhHdmiIn_ValStat_t, *puhHdmiIn_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 648

Name	Description
`uhHdmiIn_ValStatInvalid`	The signal strength property is temporarily not available.
`uhHdmiIn_ValStatKnown`	The signal strength property is available and determined.

8.1.7.2.2 Model Types & Constants

None

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.7.3 Logical Component

8.1.7.3.1 Interface-Role Model

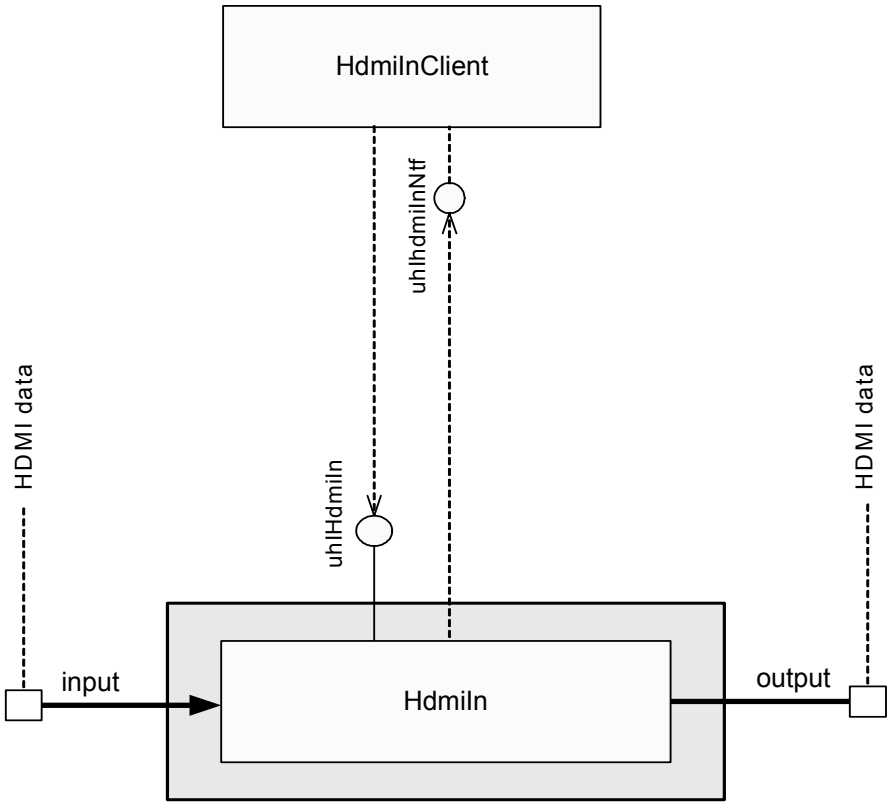


Figure 69 — Interface Role Model

Figure 69 depicts the interface-role model of the `HdmiIn`. It shows the control and notification interfaces and the roles that the `HdmiIn` and `HdmiInClient` can have. In this logical model the HDMI data is transparent. The notifications contain the data of the info frames as specified in [17] and [18]. The `HdmiIn` is represented by the `HdmiIn` role.

The `HdmiIn` provides facilities to subscribe/unsubscribe to notifications of events related to the extraction of InfoFrame data through its `UhIHdmiIn` interface. Client(s) (represented by the `HdmiInClient` role) can optionally provide one notification interface (`uhIHdmiInNtf`). This interface contains functions that will be called by the `HdmiIn` to notify events.

8.1.7.3.2 Diversity

8.1.7.3.2.1 Provided Interfaces

Table 649

Role	Interface	Presence
`HdmiIn`	`uhIHdmiIn`	mandatory
`HdmiClient`	`uhIHdmiInNtf`	optional

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.7.3.2.2 Configurable Items

None

8.1.7.3.2.3 Constraints

None

8.1.7.3.3 Instantiation

8.1.7.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 650

Type	Object	Multiplicity
`HdmiIn`	`hio`	1

8.1.7.3.3.2 Initial State

The `HdmiIn` *starts monitoring its* uncompressed digital input *stream for HDMI InfoFrames.*

The following constraints apply to the initial state of a logical component instance:

— `hio.aviInfoFrameStatus == uhHdmiIn_ValStatInvalid`
— `hio.audioInfoFrameStatus == uhHdmiIn_ValStatInvalid`
— `hio.msInfoFrameStatus == uhHdmiIn_ValStatInvalid`
— `hio.spdInfoFrameStatus == uhHdmiIn_ValStatInvalid`
— `hio.vsInfoFrameStatus == uhHdmiIn_ValStatInvalid`

8.1.7.3.4 Execution Constraints

The interface provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier “thread-safe”.

8.1.7.4 Roles

HdmiIn

Signature

```
role HdmiIn {
    ValStat                         aviInfoFrameStatus;
    ValStat                         audioInfoFrameStatus;
    ValStat                         msInfoFrameStatus;
    ValStat                         spdInfoFrameStatus;
    ValStat                         vsInfoFrameStatus;
}
```

Qualifiers

— root

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
An `HdmiIn` monitors an uncompressed digital input stream for HDMI InfoFrames.

This stream can be either HDMI or DVI, see ref [17] (subclause 8.3.3 DVI/ HDMI Device Discrimination). The `HdmiIn` can extract InfoFrames from the input stream, when present.

The `HdmiIn` provides notifications when it captured an InfoFrame for the specific type(s) the client is subscribed to. All InfoFrames being notified have been checked for validity based on their checksum.

Independent Attributes

Table 651

Name	Description
`aviInfoFrameStatus`	The validity status of the AVI Info Frame.
`audioInfoFrameStatus`	The validity status of the Audio Info Frame.
`msInfoFrameStatus`	The validity status of the MS Info Frame.
`spdInfoFrameStatus`	The validity status of the SPD Info frame.
`vsInfoFrameStatus`	The validity status of the VS Info Frame.

Dependent Attributes

None.

Invariants

None.

Instantiation
See subclause 8.1.7.3.3.

Streaming Behavior
The `HdmiIn` continually monitors its uncompressed digital input stream and performs the following actions:

Extracting InfoFrames (AVI, Audio, MS, SPD, and VS) from the HDMI stream and raising an event when it has captured an InfoFrame.

Active Behavior
The `HdmiIn` has standard notification behavior (`uhIHdmiInNtf`).

Remarks

The `HdmiIn` starts passing InfoFrame data to a client only after this client has subscribed to the notifications of the type `OnAviInfoFrame`, `OnAudioInfoFrame`, `OnMsInfoFrame` or `OnSpdInfoFrame`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.7.4.1 HdmiInClient

Signature
`role HdmiInClient {}`

Qualifiers
— actor

Description
An `HdmiInClient` represents a user of the `uhIHdmiIn` interface. No constraints apply to the behavior of the `HdmiInClient` other than those implied by Figure 69.

8.1.7.5 Interfaces

8.1.7.5.1 uhIHdmiIn

Qualifiers
— None

Description
This interface of an `HdmiIn` provides facilities to subscribe/unsubscribe to notifications of events related to the extraction of the AVI, Audio, SPD, MS and VS InfoFrames.

Interface ID
`uuid( 5560ACCB-113B-43c0-B853-87F39F51E345 )`

8.1.7.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe(
    [in] uhIHdmiInNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhHdmiIn_Notifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.7.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe(
    [in] uhIHdmiInNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhHdmiIn_Notifications_t notifs );
```

Qualifiers
— unsubscribe-function

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.7.5.2 uhlHdmiInNtf

Qualifiers
— callback

Description
This interface should be provided by each `HdmiInClient` that wants to subscribe to notifications of events related to InfoFrame data that occur in the `HdmiIn`.

Interface ID
`uuid( 721F61A0-9348-42b4-8C5C-78A028760804 )`

8.1.7.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged(
    [in] UInt32 cookie,
    [in] uhHdmiIn_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.7.5.2.2 OnAviInfoFrame

Signature
```
[async] Void OnAviInfoFrame(
    [in] UInt32 cookie,
    [in] uhHdmiIn_ValStat_t  status,
    [in] UInt8 size,
    [in, size_is(size)] const UInt8 *pBuffer );
```

Qualifiers
None

Description
This function is called by the `HdmiIn` each time it has monitored an HDMI InfoFrame of its HDMI input stream. The passed pointer points to the AVI InfoFrame of the monitored frame. The `HdmiIn` owns the storage space occupied by the data. The storage space is only valid during this function call.

Parameters

Table 652

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified AVI InfoFrame.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
size	The size in bytes of the InfoFrame_length contents + 4 (InfoFrame Type, InfoFrame_version InfoFrame_length).
pBuffer	Pointer to a size bytes buffer of valid data.

Return Values
None

Precondition
True

Action
— Any call-back-compliant action.

Postcondition
True

Remarks
— The first byte of the pBuffer corresponds to the packet header of the AVI InfoFrame.

— The layout of the bytes follows the HDMI standard, see [17] and [18].

8.1.7.5.2.3 OnAudioInfoFrame

Signature
```
[async] Void OnAudioInfoFrame(
    [in] UInt32 cookie,
    [in] uhHdmiIn_ValStat_t status,
    [in] UInt8 size,
    [in, size_is(size)] const UInt8 *pBuffer );
```

Qualifiers
None

Description
This function is called by the HdmiIn each time it has monitored an Audio InfoFrame of its HDMI input stream. The passed pointer points to the Audio InfoFrame of the monitored frame. The HdmiIn owns the storage space occupied by the data. The storage space is only valid during this function call.

Parameters

Table 653

Name	Description
cookie	The value that was passed by the client at subscription time.
status	The validity status of the notified Audio InfoFrame.
size	The size in bytes of the InfoFrame_length contents + 4 (InfoFrame Type, InfoFrame_version InfoFrame_length).
pBuffer	Pointer to a size bytes buffer of valid data.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None

Precondition
`True`

Action
— Any call-back-compliant action.

Postcondition
`True`

Remarks
— The first byte of the `pBuffer` corresponds to the packet header of the Audio InfoFrame.

— The layout of the bytes follows the HDMI standard, see [17] and [18].

8.1.7.5.2.4 OnMsInfoFrame

Signature
```
[async] Void OnMsInfoFrame(
    [in] UInt32 cookie,
    [in] uhHdmiIn_ValStat_t status,
    [in] UInt8 size,
    [in, size_is(size)] const UInt8 *pBuffer );
```

Qualifiers
None

Description
This function is called by the `HdmiIn` each time it has monitored a MS InfoFrame of its HDMI input stream. The passed pointer points to the MS InfoFrame of the monitored frame. The `HdmiIn` owns the storage space occupied by the data. The storage space is only valid during this function call.

Parameters

Table 654

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified MS Info InfoFrame.
`size`	The size in bytes of the InfoFrame_length contents + 4 (InfoFrame Type, InfoFrame_version InfoFrame_length).
`pBuffer`	Pointer to a size bytes buffer of valid data.

Return Values
None

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any call-back-compliant action.

Postcondition
`True`

Remarks
— The first byte of the `pBuffer` corresponds to the packet header of the MS InfoFrame.

— The layout of the bytes follows the HDMI standard, see [17] and [18].

8.1.7.5.2.5 OnSpdInfoFrame

Signature
```
[async] Void OnSpdInfoFrame(
    [in] UInt32 cookie,
    [in] uhHdmiIn_ValStat_t status,
    [in] UInt8 size,
    [in, size_is(size)] const UInt8 *pBuffer );
```

Qualifiers
None

Description
This function is called by the `HdmiIn` each time it has monitored an SPD InfoFrame of its HDMI input stream. The passed pointer points to the SPD InfoFrame of the monitored frame. The `HdmiIn` owns the storage space occupied by the data. The storage space is only valid during this function call.

Parameters

Table 655

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified SPD InfoFrame.
`size`	The size in bytes of the InfoFrame_length contents + 4 (InfoFrame Type, InfoFrame_version InfoFrame_length).
`pBuffer`	Pointer to a size bytes buffer of valid data.

Return Values
None

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any call-back-compliant action.

Postcondition
`True`

Remarks
— The first byte of the `pBuffer` corresponds to the packet header of the SPD InfoFrame.

— The layout of the bytes follows the HDMI standard, see [17] and [18].

8.1.7.5.2.6 OnVsInfoFrame

Signature
```
[async] Void OnVsInfoFrame(
    [in] UInt32 cookie,
    [in] uhHdmiIn_ValStat_t status,
    [in] UInt8 size,
    [in, size_is(size)] const UInt8 *pBuffer );
```

Qualifiers
None

Description
This function is called by the `HdmiIn` each time it has monitored a VS InfoFrame of its HDMI input stream. The passed pointer points to the VS InfoFrame of the monitored frame. The `HdmiIn` owns the storage space occupied by the data. The storage space is only valid during this function call.

Parameters

Table 656

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of the notified VS InfoFrame.
`size`	The size in bytes of the InfoFrame_length contents + 4 (InfoFrame Type, InfoFrame_version, InfoFrame_length).
`pBuffer`	Pointer to a size bytes buffer of valid data.

Return Values
None

Precondition
`True`

Action
— Any call-back-compliant action.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

Remarks
— The number of bytes of the VS InfoFrame is known: 31 bytes.

— The first byte of the `pBuffer` corresponds to the packet header of the VS InfoFrame.

— The layout of the bytes follows the HDMI standard, see [17] and [18].

8.1.8 UhlImageDec

8.1.8.1 UhlImageConcepts

8.1.8.1.1 Introduction

The function of an Image Decoding logical component is to decode encoded images. It can decode full images to a requested format, get thumbnails in a requested format via decoding or extraction, and extract raw meta-data. It will scale the image after decoding when the original and requested resolutions differ.

An Image Decoding logical component takes encoded image data as input and sends the decoded image to the client via a notification function. The decoder is based on a logical streaming model and can therefore process the provided input data only once for the input. When input data needs to be processed a second time, the input data has to be provided to the component anew.

The image decoder is modeled using the following processes:

— Obtainment of basic encoded image information i.e. image resolution, aspect ratio and orientation. This process can be triggered by calling the `GetOrgFullResolution` function and needs to be called before any requests to decode the main image can be made.

— Detection of thumbnail presence and obtainment of basic thumbnail image information i.e. thumbnail resolution. This process can be triggered by calling the `GetThumbnailPresent` function and needs to be called before any requests to decode or extract a thumbnail can be made.

— Main image decoding process; this process can be triggered by calling the `DecodeImage` function.

— Thumbnail image decoding/extraction process; this process can be triggered by calling the `DecodeThumbnail` function.

— Raw meta-data extraction, if this is possible depends on the image format and the policy of the platform (not all meta-data might be available for retrieval by the client as raw data). Whether raw meta-data, for retrieval by the client is present can be queried by calling the `GetMetaDataPresent` function. If this function returns True, the raw meta-data extraction process can be triggered by calling the `GetRawMetaData` function.

All the processes mentioned above when triggered wait for input data to arrive in order to start processing. How long a process will wait before indicating a failure is context/implementation dependent and not part of this specification.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.2 Types & Constants

8.1.8.2.1 Public Types & Constants

8.1.8.2.1.1 Error Codes

None.

8.1.8.2.1.2 uhImageDec2_Notification_t

Signature

```
typedef enum _uhImageDec2_Notification_t {
    uhImageDec2_SubscriptionChanged   = 0x00000001,
    uhImageDec2_FullImageDecoded      = 0x00000002,
    uhImageDec2_ThumbnailDecoded      = 0x00000004,
    uhImageDec2_MetaDataFound         = 0x00000008,
    uhImageDec2_EndMetaData           = 0x00000010,
    uhImageDec2_ResolutionObtained    = 0x00000020,
    uhImageDec2_ThumbnailPresence     = 0x00000040
} uhImageDec2_Notification_t, *puhImageDec2_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIImageDec2Ntf`.

Values

Table 657

Name	Description
`uhImageDec2_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhImageDec2_FullImageDecoded`	Represents `OnFullImageDecoded`.
`uhImageDec2_ThumbnailDecoded`	Represents `OnThumbnailDecoded`.
`uhImageDec2_MetaDataFound`	Represents `OnMetaDataFound`.
`uhImageDec2_EndMetaData`	Represents `OnEndMetaData`.
`uhImageDec2_ResolutionObtained`	Represents `OnResolutionObtained`.
`uhImageDec2_ThumbnailPresence`	Represents `OnThumbnailPresence`.

8.1.8.2.1.3 uhImageDec2_Notifications_t

Signature

```
typedef Int32 uhImageDec2_Notifications_t, *puhImageDec2_Notifications_t;
```

Qualifiers

— enum-set

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type `uhImageDec2_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints
None.

8.1.8.2.1.4 uhImageDec2_AllNotifications

Signature
```
const uhImageDec2_Notifications_t uhImageDec2_AllNotifications = 0x0000007F;
```

Qualifiers
None.

Description
Defines the set of all values of type `uhImageDec2_Notification_t`. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.1.8.2.1.5 uhImageDec2_ImageFormat_t

Signature
```
typedef enum _uhImageDec2_ImageFormat_t {
    uhImageDec2_JpegExif      = 0x00000001,
    uhImageDec2_ProgJpegExif  = 0x00000002,
    uhImageDec2_JpegJfif      = 0x00000004,
    uhImageDec2_ProgJpegJfif  = 0x00000008,
    uhImageDec2_JpegSpiff     = 0x00000010,
    uhImageDec2_ProgJpegSpiff = 0x00000020,
    uhImageDec2_Jpeg2k        = 0x00000040,
    uhImageDec2_Gif           = 0x00000080,
    uhImageDec2_InterlacedGif = 0x00000100,
    uhImageDec2_AnimatedGif   = 0x00000200,
    uhImageDec2_Bmp           = 0x00000400,
    uhImageDec2_Png           = 0x00000800,
    uhImageDec2_Tiff          = 0x00001000,
    uhImageDec2_Targa         = 0x00002000,
    uhImageDec2_Pcd           = 0x00004000,
    uhImageDec2_Jpeg          = 0x00008000,
    uhImageDec2_ProgJpeg      = 0x00010000
} uhImageDec2_ImageFormat_t, *puhImageDec2_ImageFormat_t;
```

Qualifiers
— enum-element

Description
Values of this type represent commonly used graphics content and file formats.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 658

Name	Description
uhImageDec2_JpegExif	JPEG encoded EXIF compliant content.
uhImageDec2_ProgJpegExif	Progressive JPEG encoded EXIF compliant content.
uhImageDec2_JpegJfif	JPEG encoded JFIF compliant content.
uhImageDec2_ProgJpegJfif	Progressive JPEG encoded JFIF compliant content.
uhImageDec2_JpegSpiff	JPEG encoded SPIFF compliant content.
uhImageDec2_ProgJpegSpiff	Progressive JPEG encoded SPIFF compliant content.
uhImageDec2_Jpeg2k	JPEG 2000 encoded content.
uhImageDec2_Gif	GIF encoded content.
uhImageDec2_InterlacedGif	Interlaced GIF encoded content.
uhImageDec2_AnimatedGif	Animated GIF encoded content.
uhImageDec2_Bmp	Bitmap graphics content.
uhImageDec2_Png	PNG compliant content.
uhImageDec2_Tiff	TIFF compliant content.
uhImageDec2_Targa	Truevision Targa compliant content.
uhImageDec2_Pcd	Kodak Photo CD compliant content.
uhImageDec2_Jpeg	JPEG encoded content without any file format.
uhImageDec2_ProgJpeg	Progressive JPEG encoded content without any file format.

8.1.8.2.1.6 uhImageDec2_ImageFormats_t

Signature

```
typedef Int32 uhImageDec2_ImageFormats_t, *puhImageDec2_ImageFormats_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of image formats that are typically constructed by logical OR-ing of values of type `uhImageDec2_ImageFormat_t`.

Constraints

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.2.1.7 uhImageDec2_Status_t

Signature

```
typedef enum _uhImageDec2_Status_t {
    uhImageDec2_Successful        = 1,
    uhImageDec2_Failed            = 2,
    uhImageDec2_UnsupportedFormat = 3,
    uhImageDec2_CorruptData       = 4
} uhImageDec2_Status_t, *puhImageDec2_Status_t;
```

Qualifiers
None.

Description
Values of this type represent the status of a completed action.

Values

Table 659

Name	Description
uhImageDec2_Successful	The action was successful.
uhImageDec2_Failed	The action failed.
uhImageDec2_UnsupportedFormat	The action failed; the format of the input data is not supported.
uhImageDec2_CorruptData	The action failed; the input data is corrupt.

8.1.8.2.1.8 uhImageDec2_Orientation_t

Signature

```
typedef enum _uhImageDec2_Orientation_t {
    uhImageDec2_Default                   =  1,
    uhImageDec2_90ClockWise               =  2,
    uhImageDec2_180ClockWise              =  3,
    uhImageDec2_270ClockWise              =  4,
    uhImageDec2_FlipHorizontal            =  5,
    uhImageDec2_FlipVertical              =  6,
    uhImageDec2_FlipHorizontal_90ClockWise =  7,
    uhImageDec2_FlipVertical_90ClockWise   =  8
} uhImageDec2_Orientation_t, *puhImageDec2_Orientation_t;
```

Qualifiers
None.

Description
Values of this type represent the image orientation. This type of value indicates the required rotation for the image.

Values

Table 660

Name	Description
uhImageDec2_Default	Default orientation

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhImageDec2_90ClockWise	90 degree clock wise rotation
uhImageDec2_180ClockWise	180 degree clock wise rotation
uhImageDec2_270ClockWise	270 degree clock wise rotation
uhImageDec2_FlipHorizontal	Horizontal flip
uhImageDec2_FlipVertical	Vertical flip
uhImageDec2_FlipHorizontal_90ClockWise	Flip horizontal and 90 degree clock wise rotation
uhImageDec2_FlipVertical_90ClockWise	Flip Vertical and 90 degree rotation

8.1.8.2.1.9 uhImageDec2_RotationMode_t

Signature

```
typedef enum _uhImageDec2_RotationMode_t {
    uhImageDec2_NoRotation      = 0x00000001,
    uhImageDec2_UseRotationBits = 0x00000002,
    uhImageDec2_ForcedRotation  = 0x00000004
} uhImageDec2_RotationMode_t, *puhImageDec2_RotationMode_t;
```

Qualifiers

— enum-element.

Description

Values of this type represent the image rotation mode.

Values

Table 661

Name	Description
uhImageDec2_NoRotation	No rotation to be performed. The rotation bits present in the image to be ignored.
uhImageDec2_UseRotationBits	The main image to be rotated as per the rotation bits present in the image. Thumbnail also to be rotated as per the rotation bits present in the image.
uhImageDec2_ForcedRotation	The main image to be rotated as per the rotation specified by the client. No rotation to be performed on thumbnail.

8.1.8.2.1.10 uhImageDec2_RotationModes_t

Signature

```
typedef Int32 uhImageDec2_RotationModes_t, *puhImageDec2_RotationModes_t;
```

Qualifiers

— enum-set

Description

Values of this type represent the set of rotation modes that are constructed by logical OR-ing of values of type uhImageDec2_RotationMode_t.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.2.1.11 uhImageDec2_ImageArea_t

Signature

```
typedef struct _uhImageDec2_ImageArea_t {
    UInt32 topLeftX;
    UInt32 topLeftY;
    UInt32 bottomRightX;
    UInt32 bottomRightY;
} uhImageDec2_ImageArea_t, *puhImageDec2_ImageArea_t;
```

Qualifiers
None.

Description
This type of structure represent the window of the encoded image. Typically it will be used to indicate the image area of the encoded image to be decoded.

Fields

Table 662

Name	Description
topLeftX	Top left X coordinate within the encoded image
topLeftY	Top left Y coordinate within the encoded image
bottomRightX	Bottom right X coordinate within the encoded image
bottomRightY	Bottom right Y coordinate within the encoded image

Constraints

```
( bottomRightX >= topLeftX ) AND ( topLeftY >= bottomRightY )
```

8.1.8.2.2 Model Types & Constants

None.

8.1.8.3 Logical Component

8.1.8.3.1 Interface-Role Model

Figure 70 depicts the interface-role model of the Image Decoding logical component (grey box). It shows the interfaces and the roles involved with this component.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

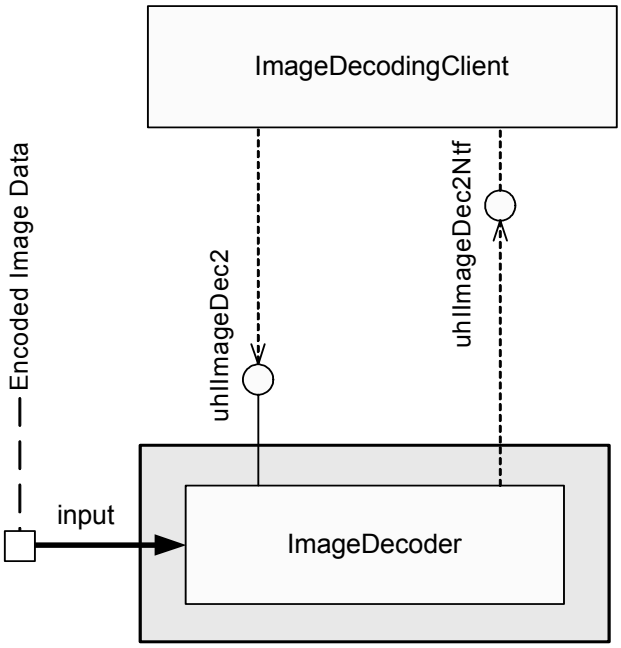


Figure 70 — Interface-Role Model

An Image Decoding logical component contains the `ImageDecoder` role that provides the control interface `uhIImageDec2`. As input, it takes encoded image data. The client can optionally provide a notification interface (`uhIImageDec2Ntf`) allowing it to receive notifications of events in the Image Decoding logical component.

8.1.8.3.2 Diversity

8.1.8.3.2.1 Provided Interfaces

Table 663

Role	Interface	Presence
`ImageDecoder`	`uhIImageDec2`	mandatory
`ImageDecodingClient`	`uhIImageDec2Ntf`	optional

8.1.8.3.2.2 Configurable Items

Table 664

Role	Attribute
`ImageDecoder`	`suppImageFormats`
`ImageDecoder`	`bufAlignment`
`ImageDecoder`	`suppPixelFormatTypes`
`ImageDecoder`	`minHorScalePercentage`
`ImageDecoder`	`minVertScalePercentage`
`ImageDecoder`	`maxHorScalePercentage`
`ImageDecoder`	`maxVertScalePercentage`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
699
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Role	Attribute
ImageDecoder	maxHorRes
ImageDecoder	maxVertRes
ImageDecoder	suppRotationModes

8.1.8.3.2.3 Constraints

```
1 <= bufAlignment
```

8.1.8.3.3 Instantiation

8.1.8.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 665

Type	Object	Multiplicity
ImageDecoder	id	1

8.1.8.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
—  id.currImageFormat == undefined
—  id.pixelFormatClass == undefined
—  id.pixelFormatType == undefined
—  id.orgHorResFullImage == 0
—  id.orgVertResFullImage == 0
—  id.thumbnailPresent == False
—  id.orgHorResThumbnail == 0
—  id.orgVertResThumbnail == 0
—  id.metaDataPresent == False
—  id.currRotationMode == uhImageDec2_NoRotation
—  id.orgImageOrientation == uhImageDec2_Default
—  id.orgImageAspectRatioNum == 0
—  id.orgImageAspectRatioDenom == 1
—  imageInfoPending == False
—  imageInfoObtained == False
—  thumbnailPresencePending == False
—  fullImagePending == False
—  thumbnailPending == False
—  metaDataPending == False
```

8.1.8.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.4 Roles

8.1.8.4.1 ImageDecoder

Signature

```
role ImageDecoder {
    const uhImageDec2_ImageFormats_t  suppImageFormats;
    uhImageDec2_ImageFormat_t         currImageFormat;
    const UInt32                      bufAlignment;
    const uhPixFmtClsSet_t            suppPixelFormatClasses;
    const uhPixFmtTypeSet_t           suppPixelFormatTypes[uhPixFmtCls_t];
    uhPixFmtCls_t                     pixelFormatClass;
    uhPixFmtType_t                    pixelFormatType;
    const UInt32                      minHorScalePercentage;
    const UInt32                      minVertScalePercentage;
    const UInt32                      maxHorScalePercentage;
    const UInt32                      maxVertScalePercentage;
    const UInt32                      maxHorRes;
    const UInt32                      maxVertRes;
    UInt32                            orgHorResFullImage;
    UInt32                            orgVertResFullImage;
    Bool                              thumbnailPresent;
    UInt32                            orgHorResThumbnail;
    UInt32                            orgVertResThumbnail;
    Bool                              metaDataPresent;
    const uhImageDec2_RotationModes_t suppRotationModes;
    uhImageDec2_RotationMode_t        currRotationMode;
    uhImageDec2_Orientation_t         orgImageOrientation;
    uhImageDec2_Orientation_t         currForcedImageOrientation;
    UInt16                            orgImageAspectRatioNum;
    UInt16                            orgImageAspectRatioDenom;
    Bool                              imageInfoPending;
    Bool                              imageInfoObtained;
    Bool                              thumbnailPresencePending;
    Bool                              fullImagePending;
    Bool                              thumbnailPending;
    Bool                              metaDataPending;
}
```

Qualifiers

— root

Description

The `ImageDecoder` role provides functionality to decode images, decode and extract thumbnails, and get raw meta-data.

Independent Attributes

Table 666

Name	Description
`suppImageFormats`	The supported image formats.
`currImageFormat`	The current image format.
`bufAlignment`	The byte alignment of a line in an image buffer. The alignment can pose restrictions on the image buffers the client can pass to the `ImageDecoder` object.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
suppPixelFormatTypes	A mapping giving the pixel format types for each pixel format class. The returned types can be used for the result of the decoding.
pixelFormatClass	The current pixel format class.
pixelFormatType	The current pixel format type.
minHorScalePercentage	The minimal horizontal scale factor in percentages.
minVertScalePercentage	The minimal vertical scale factor in percentages.
maxHorScalePercentage	The maximal horizontal scale factor in percentages.
maxVertScalePercentage	The maximal vertical scale factor in percentages.
maxHorRes	The maximal horizontal output resolution expressed in the number of pixels per line.
maxVertRes	The maximal vertical output resolution expressed in the number of lines.
orgHorResFullImage	The original horizontal resolution of the full image received via the input. The resolution is expressed in the number of pixels per line.
orgVertResFullImage	The original vertical resolution of the full image received via the input. The resolution is expressed in the number of lines.
thumbnailPresent	Indicates whether the input contains a thumbnail (True) or not (False).
orgHorResThumbnail	The original horizontal resolution of the thumbnail received via the input. The resolution is expressed in the number of pixels per line.
orgVertResThumbnail	The original vertical resolution of the thumbnail received via the input. The resolution is expressed in the number of lines.
metaDataPresent	Indicates whether raw meta-data is present (True) or not (False).
suppRotationModes	The supported rotation modes.
currRotationMode	The current rotation mode.
orgImageOrientation	The original orientation of the image received via the input.
currForcedImageOrientation	The current forced orientation of the image.
orgImageAspectRatioNum	The numerator of the aspect ratio of the original image received via the input.
orgImageAspectRatioDenom	The denominator of the aspect ratio of the original image received via the input.
imageInfoPending	Indicates whether a request for obtaining basic information about the image is pending (True) or not (False).
imageInfoObtained	Indicated whether basic information about the image has been obtained (True) or not (False).
thumbnailPresencePending	Indicates whether a request for thumbnail presence detection is pending (True) or not (False).
fullImagePending	Indicates whether a decoding request for a full image is pending (True) or not (False).
thumbnailPending	Indicates whether a decoding request for a thumbnail image is pending (True) or not (False).
metaDataPending	Indicates whether a request for extracted raw meta-data is pending (True) or not (False).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes

Table 667

Name	Description
`suppPixelFormatClasses`	The supported pixel format classes for decoded images: a class c is in `suppPixelFormatClasses` if and only if `suppPixelFormatTypes[c]` is not empty.

Invariants

— `1 <= bufAlignment`
— `orgImageAspectRatioDenom > 0`

Instantiation
See subclause 8.1.8.3.3

Streaming Behavior
Encoded image data is streamed into the `ImageDecoder`. The `ImageDecoder` decodes the data and sends the result to the client via a notification function. It has no streaming output.

Active Behavior
The `ImageDecoder` has standard notification behavior (see `uhIImageDec2Ntf`).

8.1.8.4.2 ImageDecodingClient

Signature
```
role ImageDecodingClient{}
```

Qualifiers
— actor

Description
An `ImageDecodingClient` represents a user of the `uhIImageDec2` interface. No constraints apply to the behavior of the `ImageDecodingClient` other than those implied by Figure 70.

8.1.8.5 Interfaces

8.1.8.5.1 uhIImageDec2

Qualifiers
— None

Description
This interface provides functionality to decode images, decode and extract thumbnails, and get raw meta-data.

Interface ID
`uuid( 8F71C7D0-768E-4444-9FC3-2CC63D595861 )`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIImageDec2Ntf *pINotify,
    [in] UInt32 cookie,
    [in] uhImageDec2_Notifications_t notifs );
```

Qualifiers

— subscribe-function

8.1.8.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIImageDec2Ntf *pINotify,
    [in] UInt32 cookie,
    [in] uhImageDec2_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.8.5.1.3 GetSuppImageFormats

Signature

```
uhErrorCode_t GetSuppImageFormats (
    [out] puhImageDec2_ImageFormats_t pFormats );
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns the supported image formats. The returned integer value represents the OR-ed supported image formats.

Parameters

Table 668

Name	Description
`pFormats`	Output parameter returning the supported image formats.

Return Values

Standard.

Precondition

`True`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
`*pFormats == id.suppImageFormats`

8.1.8.5.1.4 GetBufAlignment

Signature
```
uhErrorCode_t GetBufAlignment ( [out] pUInt32 pBufAlignment );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the byte alignment required of image buffer lines. The byte alignment poses restrictions on the stride the client specifies for image buffers (see `DecodeImage` and `DecodeThumbnail`).

Parameters

Table 669

Name	Description
`pBufAlignment`	Output parameter returning the byte alignment of the lines of an image buffer.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pBufAlignment == id.bufAlignment`

8.1.8.5.1.5 GetSuppFormatClasses

Signature
```
uhErrorCode_t GetSuppFormatClasses ( [out] puhPixFmtClsSet_t pFormatClasses );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the supported format classes to which an image can be decoded. The returned integer value represents the OR-ed supported format classes.

Parameters

Table 670

Name	Description
`pFormatClasses`	Output parameter returning the supported format classes to which an image can be decoded.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pFormatClasses == id.suppPixelFormatClasses`

8.1.8.5.1.6 GetSuppFormatTypes

Signature
```
uhErrorCode_t GetSuppFormatTypes (
    [in] uhPixFmtCls_t formatClass,
    [out] puhPixFmtTypeSet_t pFormatTypes );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the supported format types of the specified pixel format class to which an image can be decoded. The returned integer value represents the OR-ed supported format types.

Parameters

Table 671

Name	Description
`formatClass`	The pixel class for which the supported pixel format types are requested.
`pFormatTypes`	Output parameter returning the supported pixel format types for the pixel format class `formatClass`.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
— `formatClass` in `id.suppPixelFormatClasses`

Action
None.

Postcondition
`*pFormatTypes == id.suppPixelFormatTypes[formatClass]`

8.1.8.5.1.7 GetScaleRange

Signature
```
uhErrorCode_t GetScaleRange (
    [out] pUInt32 pMinHorPercentage,
    [out] pUInt32 pMinVertPercentage,
    [out] pUInt32 pMaxHorPercentage,
    [out] pUInt32 pMaxVertPercentage );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the minimal and maximal horizontal and vertical scale factors in percentages that images can be scaled to after decoding. A number higher than 100 indicates up-scaling, equal to 100 indicates no scaling, and lower than 100 indicates down scaling.

Parameters

Table 672

Name	Description
`pMinHorPercentage`	Output parameter returning the minimal horizontal scale factor in percentages.
`pMinVertPercentage`	Output parameter returning the minimal vertical scale factor in percentages.
`pMaxHorPercentage`	Output parameter returning the maximal horizontal scale factor in percentages.
`pMaxVertPercentage`	Output parameter returning the maximal vertical scale factor in percentages.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMinHorPercentage == id.minHorScalePercentage`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— *pMinVertPercentage == id.minVertScalePercentage
— *pMaxHorPercentage == id.maxHorScalePercentage
— *pMaxVertPercentage == id.maxVertScalePercentage

8.1.8.5.1.8 GetMaxResolution

Signature
```
uhErrorCode_t GetMaxResolution (
    [out] pUInt32 pMaxHorRes,
    [out] pUInt32 pMaxVertRes );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the maximal horizontal and vertical resolution that images can be decoded to.

Parameters

Table 673

Name	Description
pMaxHorRes	Output parameter returning the maximal horizontal resolution.
pMaxVertRes	Output parameter returning the maximal vertical resolution.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pMaxHorRes == id.maxHorRes
— *pMaxVertRes == id.maxVertRes

8.1.8.5.1.9 GetOrgFullResolution

Signature
```
uhErrorCode_t GetOrgFullResolution ( );
```

Qualifiers
— asynchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
This function triggers the retrieval of the original resolution of the full image. Before calling this function, the client needs to be subscribed to `OnResolutionObtained`.

Parameters
None.

Return Values
Standard.

Precondition
`id.imageInfoPending == False`

Action
— Modify `id.imageInfoPending`

Postcondition
— `id.imageInfoPending == True`

Asynchronous Action
— Modify `id.imageInfoPending`
— Modify `id.imageInfoObtained`
— Modify `id.orgHorResFullImage`
— Modify `id.orgVertResFullImage`
— Notify `OnResolutionObtained`

Asynchronous Postcondition
— `id.imageInfoPending == False`
— if the resolution has been obtained
  — `id.imageInfoObtained == True`
  — `id.orgHorResFullImage != 0`
  — `id.orgVertResFullImage != 0`
— else
  — `id.imageInfoObtained == False`
  — `id.orgHorResFullImage == 0`
  — `id.orgVertResFullImage == 0`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.5.1.10 DecodeImage

Signature

```
uhErrorCode_t DecodeImage (
    [in] UInt32 requestedHorRes,
    [in] UInt32 requestedVertRes,
    [in] UInt32 stride,
    [in] uhPixFmtCls_t formatClass,
    [in] uhPixFmtType_t formatType,
    [in] Bool useImageArea,
    [in] uhImageDec2_ImageArea_t requestedImageArea,
    [in] UInt32 bufferSize,
    [out, size_is(bufferSize)] pUInt8 pBuffer );
```

Qualifiers

— asynchronous

— single-threaded

Description

This function requests the decoding of an image in the requested format and resolution. A partial area of the encoded image may be selected for decoding. The coordinates of the partial area are with respect to the encoded image. When the requested resolution differs from the resolution of the encoded image, or from the resolution of the requested encoded image area, the image is scaled to the requested resolution. It is the client's responsibility to assure that the resulting scale factor including the effects of applied rotation (see `SetRotationMode`) is within the supported range (see `GetScaleRange`). There is no support for pixel formats requiring multiple buffers (e.g. one for Y, one for UV).

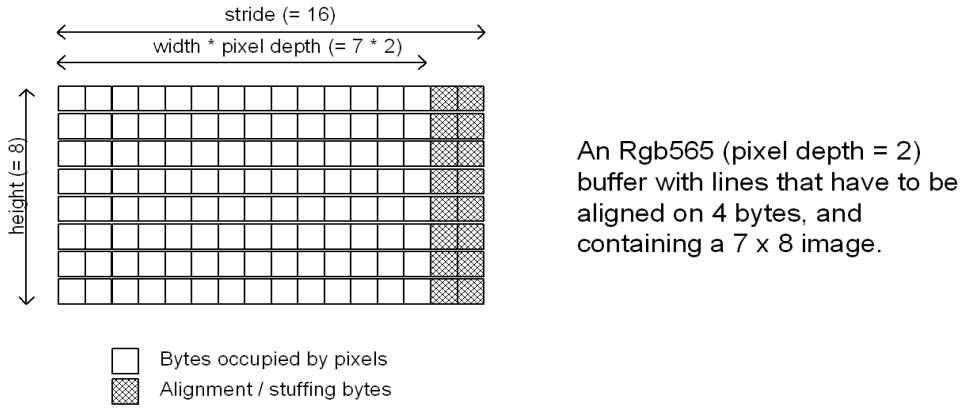


Figure 71 — An example showing the relation between pixel type, image size, buffer size and stride.

Parameters

Table 674

Name	Description
`requestedHorRes`	The requested horizontal resolution (width in Figure 71) of the returned image in number of pixels per line.
`requestedVertRes`	The requested vertical resolution (height in Figure 71) of the returned image in number of lines.
`stride`	The stride (in bytes) of the image. See Figure 71.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
formatClass	The requested pixel format class.
formatType	The requested pixel format type.
useImageArea	Flag indicating if the encoded image should be partially decoded (True) or not (False). The area of the encoded image to be decoded is indicated by requestedImageArea.
requestedImageArea	The area of the encoded image to be decoded if useImageArea equals True.
bufferSize	The size of the buffer where image is to be decoded in bytes.
pBuffer	Pointer to the first element of the client's buffer where image is to be decoded.

Return Values
Standard.

Precondition

```
— id.fullImagePending == False
— requestedHorRes <= id.maxHorRes
— requestedVertRes <= id.maxVertRes
— stride >= requestedHorRes * pixel depth of formatType
— stride mod id.bufAlignment == 0
— bufferSize == stride * requestedVertRes
— formatClass in id.suppPixelFormatClasses (see function GetSuppFormatClasses)
— formatType in id.suppPixelFormatTypes[formatClass] (see function
  GetSuppFormatTypes)
— if ( useImageArea == True )
  — 0 <= requestedImageArea.topLeftX < id.orgHorResFullImage
  — 0 <= requestedImageArea.topLeftY < id.orgVertResFullImage
  — 0 < requestedImageArea.bottomRightX <= id.orgHorResFullImage
  — 0 < requestedImageArea.bottomRightY <= id.orgVertResFullImage
— if ( useImageArea == True )
  — let horInputRes = requestedImageArea.bottomRightX -
    requestedImageArea.topLeftX
  — let vertInputRes = requestedImageArea.bottomRightY -
    requestedImageArea.topLeftY
— else
  — let horInputRes = id.orgHorResFullImage
  — let vertInputRes = id.orgVertResFullImage
— let rotDeg = the applied rotation in degrees
— if ( rotDeg == 0 modulo 180 )
  — id.minHorScalePercentage <= ( requestedHorRes / horInputRes ) * 100 <=
    id.maxHorScalePercentage
  — id.minVertScalePercentage <= ( requestedVertRes / vertInputRes ) * 100 <=
    id.maxVertScalePercentage
— if ( rotDeg == 90 modulo 180 )
  — id.minHorScalePercentage <= ( requestedVertRes / horInputRes ) * 100 <=
    id.maxHorScalePercentage
  — id.minVertScalePercentage <= ( requestedHorRes / vertInputRes ) * 100 <=
    id.maxVertScalePercentage
```

Action
Modify id.fullImagePending

Postcondition

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
id.fullImagePending == True
```

Asynchronous Action
- Modify `id.fullImagePending`
- Notify `OnFullImageDecoded`

Asynchronous Postcondition
- `id.fullImagePending == False`
- If decoding was successful
  - `pBuffer` points to the decoded full image.

8.1.8.5.1.11 GetThumbnailPresent

Signature
```
uhErrorCode_t GetThumbnailPresent ( );
```

Qualifiers
- asynchronous
- thread-safe

Description
This function triggers the detection of thumbnail presence. Before calling this function, the client needs to be subscribed to `OnThumbnailPresence`.

Parameters
None.

Return Values
Standard.

Precondition
```
id.thumbnailPresencePending == False
```

Action
- Modify `id.thumbnailPresencePending`

Postcondition
```
id.thumbnailPresencePending == True
```

Asynchronous Action
- Modify `id.thumbnailPresencePending`
- Modify `id.thumbnailPresent`
- Notify `OnThumbnailPresence`

Asynchronous Postcondition
- `id.thumbnailPresencePending == False`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— if `thumbnail` is present
  — `id.thumbnailPresent == True`
— else
  — `id.thumbnailPresent == False`

8.1.8.5.1.12 GetOrgThumbnailResolution

Signature

```
uhErrorCode_t GetOrgThumbnailResolution (
    [out] pUInt32 pHorRes,
    [out] pUInt32 pVertRes );
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns the original resolution of the thumbnail.

Parameters

Table 675

Name	Description
`pHorRes`	Output parameter returning the original horizontal resolution of the thumbnail received via the input. The resolution is expressed in number of pixels per line.
`pVertRes`	Output parameter returning the original vertical resolution of the thumbnail received via the input. The resolution is expressed in number of lines.

Return Values

Standard.

Precondition

`id.thumbnailPresent == True`

Action

None.

Postcondition

— `*pHorRes == id.orgHorResThumbnail`
— `*pVertRes == id.orgVertResThumbnail`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.5.1.13 DecodeThumbnail

Signature

```
uhErrorCode_t DecodeThumbnail (
    [in] UInt32 requestedHorRes,
    [in] UInt32 requestedVertRes,
    [in] UInt32 stride,
    [in] uhPixFmtCls_t formatClass,
    [in] uhPixFmtType_t formatType,
    [in] UInt32 bufferSize,
    [out, size_is(bufferSize)] pUInt8 pBuffer );
```

Qualifiers

— asynchronous

— single-threaded

Description

This function requests the decoding or extraction of a thumbnail in the requested format and resolution. When the requested resolution differs from the resolution of the encoded image, the image is scaled to the requested resolution. It is the client's responsibility to assure that the resulting scale factor including the effects of applied rotation (see `SetRotationMode`) is within the supported range (see `GetScaleRange`). There is no support for pixel formats requiring multiple buffers (e.g. one for Y, one for UV).

Parameters

Table 676

Name	Description
`requestedHorRes`	The requested horizontal resolution (width in Figure 71) of the returned image in number of pixels per line.
`requestedVertRes`	The requested vertical resolution (height in Figure 71) of the returned image in number of lines.
`stride`	The stride (in bytes) of the thumbnail. See Figure 71.
`formatClass`	The requested pixel format class.
`formatType`	The requested pixel format type.
`bufferSize`	The size of the buffer where thumbnail is to be decoded or extracted in bytes.
`pBuffer`	Pointer to the first element of the client's buffer where thumbnail is to be decoded or extracted.

Return Values

Standard.

Precondition

— `id.thumbnailPending == False`
— `id.thumbnailPresent == True`
— `requestedHorRes <= id.maxHorRes`
— `requestedVertRes <= id.maxVertRes`
— `stride >= requestedHorRes *` pixel depth of `formatType`
— `stride` mod `id.bufAlignment == 0`
— `bufferSize == stride * requestedVertRes`
— `formatClass` in `id.suppPixelFormatClasses` (see function `GetSuppFormatClasses`)

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— formatType in id.suppPixelFormatTypes[formatClass] (see function GetSuppFormatTypes)
— let rotDeg = the applied rotation in degrees
— if (rotDeg == 0 modulo 180)
  — id.minHorScalePercentage <= (requestedHorRes / id.orgHorResThumbnail) * 100 <= id.maxHorScalePercentage
  — id.minVertScalePercentage <= (requestedVertRes / id.orgVertResThumbnail) * 100 <= id.maxVertScalePercentage
— if (rotDeg == 90 modulo 180)
  — id.minHorScalePercentage <= (requestedVertRes / id.orgHorResThumbnail) * 100 <= id.maxHorScalePercentage
  — id.minVertScalePercentage <= (requestedHorRes / id.orgVertResThumbnail) * 100 <= id.maxVertScalePercentage

Action
Modify id.thumbnailPending

Postcondition

id.thumbnailPending == True

Asynchronous Action
— Modify id.thumbnailPending
— Notify OnThumbnailDecoded

Asynchronous Postcondition
— id.thumbnailPending == False
— if decoding or extraction was successful
  — pBuffer points to the decoded or extracted thumbnail.

8.1.8.5.1.14 GetMetaDataPresent

Signature
```
uhErrorCode_t GetMetaDataPresent (
    [out] pBool pMetaDataPresent );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns whether meta-data is present that can be retrieved by the client as raw meta-data. When meta-data is present, the raw meta-data can be retrieved by subscribing to OnMetaDataFound and OnEndMetaData and calling GetRawMetaData.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 677

Name	Description
pMetaDataPresent	Output parameter returning whether extracted raw meta-data is present (True) or not (False).

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pMetaDataPresent == id.metaDataPresent

8.1.8.5.1.15 GetRawMetaData

Signature
```
uhErrorCode_t GetRawMetaData (
    [out] puhImageDec2_ImageFormat_t pImageFormat );
```

Qualifiers
- asynchronous
- single-threaded

Description
This function requests the retrieval of raw meta-data. Before calling this function, a client needs to check if meta-data is present by calling GetMetaDataPresent. To receive the meta-data, the client has to subscribe to uhImageDec2_MetaDataFound and uhImageDec2_EndMetaData. After calling this function, extracted meta-data will be notified by calling the OnMetaDataFound function of the client one or more times. When no more meta-data is present, the OnEndMetaData function of the client will be called.

Parameters

Table 678

Name	Description
pImageFormat	Output parameter indicating the format of the image.

Return Values
Standard.

Precondition
- id.metaDataPending == False
- id.metaDataPresent == True

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `id.metaDataPending`

Postcondition
— `id.metaDataPending == True`
— `*pImageFormat in suppImageFormats`

Asynchronous Action
— if no more meta-data will be notified
  — Modify `id.metaDataPending`
  — Notify `OnMetaDataFound`
— else
  — Notify `OnMetaDataFound`

Asynchronous Postcondition
— if no more meta-data will be notified

`id.metaDataPending == False`

8.1.8.5.1.16 GetOrgImageOrientation

Signature
```
uhErrorCode_t GetOrgImageOrientation (
    [out] puhImageDec2_Orientation_t pOrientation );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the orientation of the image as described in the rotation bits of the image.

Parameters

Table 679

Name	Description
`pOrientation`	Output parameter providing the orientation of the image.

Return Values
Standard.

Precondition
`id.imageInfoObtained == True`

Action
None.

Postcondition
`*pOrientation == id.orgImageOrientation`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.5.1.17 GetOrgImageAspectRatio

Signature
```
uhErrorCode_t GetOrgImageAspectRatio(
    [out] pUInt16 pNumerator,
    [out] pUInt16 pDenominator );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the aspect ratio of the original image.

Parameters

Table 680

Name	Description
`pNumerator`	Output parameter returning the numerator of the aspect ratio of the input.
`pDenominator`	Output parameter returning the denominator of the aspect ratio of the input.

Return Values
Standard.

Precondition
```
id.imageInfoObtained == True
```

Action
None.

Postcondition
— `*pNumerator == id.orgImageAspectRatioNum`
— `*pDenominator == id.orgImageAspectRatioDenom`

8.1.8.5.1.18 GetSupportedRotationModes

Signature
```
uhErrorCode_t GetSupportedRotationModes (
    [out] puhImageDec2_RotationModes_t pModes );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the rotation modes supported by the Image Decoder. The returned integer value represents the OR-ed supported rotation modes.

Parameters

Table 681

Name	Description
pModes	Output parameter providing rotation modes supported by the image decoder.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pModes == id.suppRotationModes

8.1.8.5.1.19 SetRotationMode

Signature
```
uhErrorCode_t SetRotationMode (
    [in] uhImageDec2_RotationMode_t mode,
    [in] uhImageDec2_Orientation_t orientation );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the rotation mode. The rotation mode will be used for decoding the main image and the thumbnail. If the mode parameter is set to forced rotation mode, the orientation parameter specifies the orientation for the rotation. For all other rotation modes the orientation parameter is ignored.

Parameters

Table 682

Name	Description
mode	The rotation mode to be used
orientation	The orientation to be used if the rotation mode specifies forced rotation.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
— `mode` in `id.suppRotationModes`

Action
— Modify `id.currRotationMode`
— `Modify id.currForcedImageOrientation`

Postcondition
— `id.currRotationMode == mode`
— `id.currForcedImageOrientation == orientation`

8.1.8.5.1.20 GetRotationMode

Signature
```
uhErrorCode_t GetRotationMode (
    [out] puhImageDec2_RotationMode_t pMode,
    [out] puhImageDec2_Orientation_t pOrientation );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the rotation mode used for decoding the main image and the thumbnail.

Parameters

Table 683

Name	Description
`pMode`	The rotation mode that is currently set.
`pOrientation`	The orientation that is currently set. This orientation is only applied if the rotation mode specifies forced rotation.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMode == id.currRotationMode`
— `*pOrientation == id.currForcedImageOrientation`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.5.2 uhIImageDec2Ntf

Qualifiers
— callback

Description
This is a notification interface to be provided by clients of the Image Decoding logical component that wish to receive notifications of Image Decoding related events.

Interface ID
`uuid( A182567C-FAA3-48d3-A5A6-2CD827893203 )`

8.1.8.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhImageDec2_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.8.5.2.2 OnFullImageDecoded

Signature
```
[async] Void OnFullImageDecoded (
    [in] UInt32 cookie,
    [in] uhImageDec2_Status_t status );
```

Qualifiers
None.

Description
This function is called by the `ImageDecoder` when it has either successfully decoded a full image or failed to do so. Failure to decode is reported via the `status` parameter. On success, the decoded image is present in the buffer passed with `DecodeImage`. Otherwise, this buffer is left as it was.

Parameters

Table 684

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The status of the image decoding.

Return Values
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.8.5.2.3 OnThumbnailDecoded

Signature
```
[async] Void OnThumbnailDecoded (
    [in] UInt32 cookie,
    [in] uhImageDec2_Status_t status );
```

Qualifiers
None.

Description
This function is called by the `ImageDecoder` when it has either decoded or extracted a thumbnail or failed to do so. Failure to decode or extract is reported via the `status` parameter. On success, the decoded or extracted thumbnail is present in the buffer passed with `DecodeThumbnail`. Otherwise, this buffer is left as it was.

Parameters

Table 685

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The status of the thumbnail decoding.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.8.5.2.4 OnMetaDataFound

Signature

```
Void OnMetaDataFound (
    [in] UInt32 cookie,
    [in] UInt32 size,
    [in, size_is(size)] const pUInt8 pRawData );
```

Qualifiers
None.

Description
This function is called by the `ImageDecoder` when it has found raw meta-data in the input stream. The format of the raw meta-data is compliant with the specification of the image type and its file format. The client is expected to copy the meta-data from the buffer pointed to by `pRawData`. When this function returns the buffer pointed to by `pRawData` will no longer be valid.

Parameters

Table 686

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`size`	The size of the buffer containing the meta-data.
`pRawData`	Pointer to the buffer of the image decoder containing the raw meta-data.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.8.5.2.5 OnEndMetaData

Signature

```
[async] Void OnEndMetaData (
    [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the `ImageDecoder` when no more meta-data is available in the input stream.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 687

Name	Description
cookie	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

8.1.8.5.2.6 OnResolutionObtained

Signature

```
[async] Void OnResolutionObtained (
    [in] UInt32 cookie,
    [in] uhImageDec2_Status_t status,
    [in] UInt32 horRes,
    [in] UInt32 vertRes );
```

Qualifiers
None.

Description
This function is called by the ImageDecoder when the process to determine the resolution, aspect ratio and orientation of the encoded image in the input stream has finished.

Parameters

Table 688

Name	Description
cookie	The value that was passed by the client at subscription time.
status	The status of the resolution obtainment.
horRes	Parameter indicating the horizontal resolution. The resolution is expressed in number of pixels per line.
vertRes	Parameter indicating the vertical resolution. The resolution is expressed in number of lines.

Return Values
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.8.5.2.7 OnThumbnailPresence

Signature
```
[async] Void OnThumbnailPresence (
    [in] UInt32 cookie,
    [in] uhImageDec2_Status_t status,
    [in] Bool presence );
```

Qualifiers
None.

Description
This function is called by the `ImageDecoder` when the process to detect the presence of a thumbnail in the input stream has finished.

Parameters

Table 689

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The status of the thumbnail presence detection.
`presence`	Parameter indicating whether a thumbnail is present (`True`) or not (`False`).

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.9 UhlOob

8.1.9.1 Concepts

8.1.9.1.1 Introduction

The function of an OOBTD is to receive an external out of band signal, tune to the correct frequency and demodulate that signal.

The out of band signal is a separate signal with a small frequency bandwidth that is used to send metadata that is applicable to multiple channels in the in-band signal. Out of band tuning and demodulation is typically much faster than in-band tuning and demodulation.

This specification is typically used by clients that adhere to the OpenCable POD host control interface standard.

8.1.9.2 Types & Constants

8.1.9.2.1 Public Types & Constants

8.1.9.2.1.1 Error Codes

None.

8.1.9.2.1.2 uhOob_DataRate_t

Signature

```
typedef enum _uhOob_DataRate_t {
    uhOob_DataRateUnknown            = 0,
    uhOob_DataRateDvs167_1544        = 1,
    uhOob_DataRateDvs178_2048        = 2,
    uhOob_DataRateDvs167_3088        = 3
} uhOob_DataRate_t, *puhOob_DataRate_t;
```

Qualifiers

— enum

Description

Values of this type represent the receiver data rates according to the DVS167 and DVS178 standards (OpenCable POD host control interface standards on OOB reception).

Values

Table 690

Name	Description
uhOob_DataRateUnknown	The data rate is not known (initial value).
uhOob_DataRateDvs167_1544	OpenCable POD DVS167 1544 rate.
uhOob_DataRateDvs178_2048	OpenCable POD DVS178 2048 rate.
uhOob_DataRateDvs167_3088	OpenCable POD DVS167 3088 rate.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.9.3 Logical Component

8.1.9.3.1 Interface-Role Model

Figure 72 Depicts the interface-role model of the OOBTD logical component. It shows the control interfaces and the roles involved with this logical component.

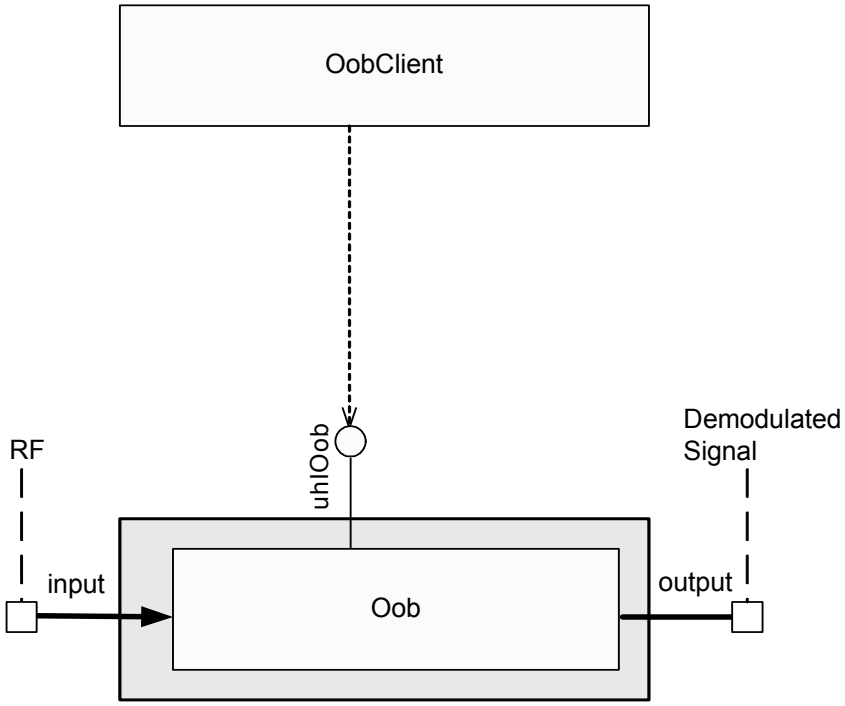


Figure 72 — Interface-Role Model

The grey box in Figure 72 denotes an OOBTD logical component containing the Oob role providing one mandatory (solid line) control interfaces to its clients.

8.1.9.3.2 Diversity

8.1.9.3.2.1 Provided Interfaces

Table 691

Role	Interface	Presence
Oob	`uhI`Oob	Mandatory

8.1.9.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of the logical component. For a description of the attributes see Roles subclause, and for how they are initialized see subclause 8.1.9.3.3.

Table 692

Role	Attribute
Oob	minFrequency

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Role	Attribute
Oob	maxFrequency

8.1.9.3.2.3 Constraints

See the InvariantsT of the Oob role (subclause 8.1.9.4.1).

8.1.9.3.3 Instantiation

8.1.9.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 693

Type	Object	Multiplicity
Oob	oob	1

8.1.9.3.3.2 Initial State

```
— Toob.setFrequency             == oob.minFrequency
— oob.setDataRate              == uhOob_DataRateUnknown
— oob.spectralInversionEnabled == False
```

8.1.9.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier “thread-safe”.

8.1.9.4 Roles

8.1.9.4.1 Oob

Signature

```
role Oob {
    const UInt32             minFrequency;
    const UInt32             maxFrequency;
    UInt32                   setFrequency;
    uhOob_DataRate_t         actualTvSystem;
    Bool                     spectralInversionEnabled;
}
```

Qualifiers

— root

Description

A Oob controls the tuning and demodulation attributes.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 694

Name	Description
`minFrequency`	The minimum OOB frequency.
`maxFrequency`	The maximum OOB frequency
`setFrequency`	The current OOB frequency.
`setDataRate`	The current OOB data rate.
`spectralInversionEnabled`	Indicates if spectral inversion is enabled (`True`) or not (`False`)

Invariants

— `oob.minFrequency ≤ oob.setFrequency ≤ oob.maxFrequency`

Instantiation

A Oob role is always instantiated as part of an OOBTD logical component; see subclause 8.1.9.3.3.

Streaming Behavior

The Oob role tunes to the correct frequency in the incoming signal, demodulates that signal, and outputs the result.

8.1.9.4.2 OobClient

Signature

```
role OobClient {}
```

Qualifiers

— actor

Description

An `OobClient` role represents a user of the control interfaces offered by the Oob Trole. No constraints apply to the behavior of the `OobClient` role other than those implied by Figure 72.

8.1.9.5 Interfaces

8.1.9.5.1 uhIOob

Qualifiers

— None.

Description

This interface of the Oob role provides control of the out of band tuning and demodulation parameters.

Interface ID

```
uuid( E657A803-C4CD-403b-9EC7-4E696FE6E290 )
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks

In the following function specifications, the name `oob` will be used to refer to the provider of this interface.

8.1.9.5.1.1 GetFrequencyRange

Signature

```
uhErrorCode_t GetFrequencyRange (
    [out] pUInt32 pMinFreq,
    [out] pUInt32 pMaxFreq );
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns the boundaries for the out of band frequency in Hz.

Parameters

Table 695

Name	Description
`pMinFreq`	Output parameter returning the minimum OOB frequency in Hz.
`pMaxFreq`	Output parameter returning the maximum OOB frequency in Hz.

Return Values

Standard.

Precondition

`True`

Action

— None.

Postcondition

— `*pMinFreq == oob.minFrequency`
— `*pMaxFreq == oob.maxFrequency`

8.1.9.5.1.2 SetFrequency

Signature

```
uhErrorCode_t SetFrequency ( [in] UInt32 freq );
```

Qualifiers

— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function sets the out of band frequency in Hz.

Parameters

Table 696

Name	Description
`freq`	Input parameter indicating the new OOB frequency in Hz.

Return Values
Standard.

Precondition
`oob.minFrequency ≤ freq ≤ oob.maxFrequency`

Action
— Modify `oob.setFrequency`

Postcondition
`oob.setFrequency == freq`

8.1.9.5.1.3 GetFrequency

Signature
```
uhErrorCode_t GetFrequency ( [out] pUInt32 pFreq );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the out of band frequency in Hz as it was set via the `SetFrequency` function.

Parameters

Table 697

Name	Description
`pFreq`	Output parameter returning the set out of band frequency in Hz.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— None.

Postcondition
`*pFreq == oob.setFrequency`

8.1.9.5.1.4 SetDataRate

Signature
```
uhErrorCode_t SetDataRate ( [in] uhOob_DataRate_t rate );
```

Qualifiers
— synchronous

— single-threaded

Description
This function sets the out of band data rate.

Parameters

Table 698

Name	Description
`rate`	Input parameter indicating the new data rate.

Return Values
Standard.

Precondition
`rate != uhOob_DataRateUnknown`

Action
— Modify `oob.actualTvSystem`

Postcondition
`oob.actualTvSystem == rate`

8.1.9.5.1.5 GetDataRate

Signature
```
uhErrorCode_t GetDataRate ( [out] puhOob_DataRate_t pRate );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the out of band data rate, as it was set via the `SetDataRate` function.

Parameters

Table 699

Name	Description
`pRate`	Output parameter returning the out of band data rate that was set.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
`*pRate == oob.actualTvSystem`

8.1.9.5.1.6 EnableSpectralInversion

Signature
```
uhErrorCode_t EnableSpectralInversion ( [in] Bool enable );
```

Qualifiers
— synchronous

— single-threaded

Description
This function enables (`True`) or disables (`False`) spectral inversion.

Parameters

Table 700

Name	Description
`enable`	Input parameter indicating if spectral inversion has to be enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `oob.spectralInversionEnabled`

Postcondition
`oob.spectralInversionEnabled == enable`

8.1.9.5.1.7 GetSpectralInversionEnabled

Signature
```
uhErrorCode_t GetSpectralInversionEnabled ( [out] pBool pEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns whether or not spectral inversion is currently enabled.

Parameters

Table 701

Name	Description
`pEnabled`	Output parameter indicating if spectral inversion is currently enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
`*pEnabled == oob.spectralInversionEnabled`

8.1.10 UhlRfAmp

8.1.10.1 Concepts

8.1.10.1.1 Introduction

The function of an RFA is to control the amplification of an incoming RF signal in order to optimize the signal processing by a tuner. Pre-amplification or attenuation is typically needed in regions where transmissions are weak, over modulated, or heavily distorted.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.10.2 Types & Constants

8.1.10.2.1 Public Types & Constants

None

8.1.10.2.2 Model Types & Constants

None

8.1.10.3 Logical Component

8.1.10.3.1 Interface-Role Model

Figure 73_depicts the interface-role model of the RFA. It shows the control interface and the roles involved with this logical component.

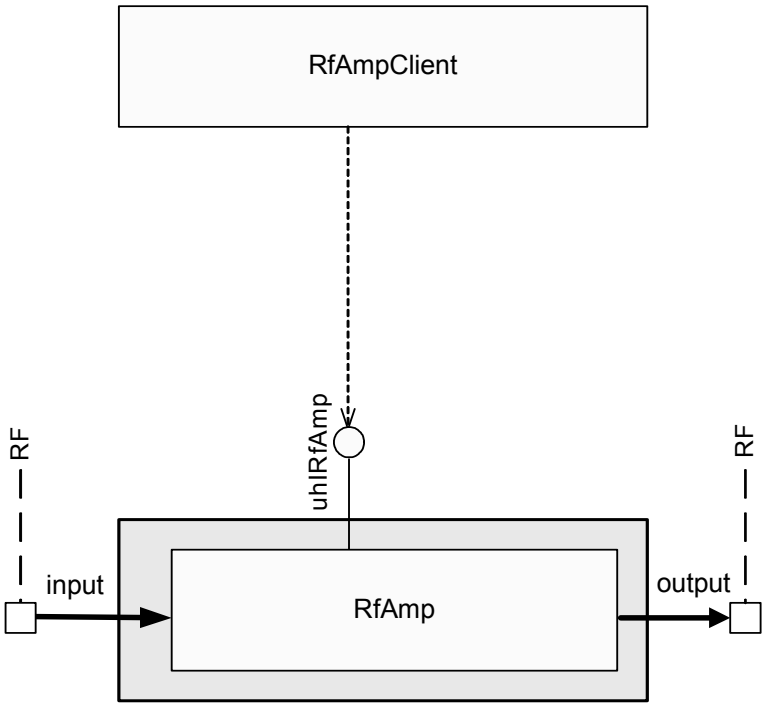


Figure 73 — Interface Role Model

The grey box in Figure 73 denotes an RFA logical component containing the RfAmp role providing one mandatory (solid line) control interface to the `RfAmpClient`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.10.3.2 Diversity

8.1.10.3.2.1 Provided Interfaces

Table 702

Role	Interface	Presence
RfAmp	uhIRfAmp	mandatory

8.1.10.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of the logical component. For a description of the attributes see the Roles subclause, and for how they are initialized, see subclause 8.1.10.3.3.

Table 703

Role	Attribute
RfAmp	minAmp
RfAmp	maxAmp

8.1.10.3.2.3 Constraints

— See the Invariants of the RfAmp role (subclause 8.1.10.4.1).

8.1.10.3.3 Instantiation

8.1.10.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 704

Type	Object	Multiplicity
RfAmp	ra	1

8.1.10.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
ra.currentAmp == 0
```

8.1.10.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.10.4 Roles

8.1.10.4.1 RfAmp

Signature

```
role RfAmp  {
    const  Int32 minAmp;
    const  Int32 maxAmp;
    Int32 currentAmp;
}
```

Qualifiers

— root

Description

An `RfAmp` controls the RF amplification attributes.

Independent Attributes

Table 705

Name	Description
`minAmp`	Minimum amplification level.
`maxAmp`	Maximum amplification level.
`currentAmp`	Current amplification level.

Dependent Attributes

None.

Invariants

— `minAmp ≤ currentAmp ≤ maxAmp`
— `minAmp ≤ 0`
— `maxAmp ≥ 0`

Instantiation

An `RfAmp` role is always instantiated as part of an RFA; see subclause 8.1.10.3.3.

Streaming Behavior

The `RfAmp` role amplifies or attenuates the incoming RF signal so an RF signal with an acceptable level can be sent to the tuner.

8.1.10.4.2 RfAmpClient

Signature

```
role RfAmpClient {}
```

Qualifiers

— actor

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
An `RfAmpClient` role represents a user of the control interfaces offered by the `RfAmp` role. No constraints apply to the behavior of the `RfAmpClient` role other than those implied by Figure 73.

8.1.10.5 Interfaces

8.1.10.5.1 uhIRfAmp

Qualifiers
— None.

Description
This interface of an `RfAmp` role provides control of the RF amplification parameters.

Interface ID
`uuid( BF703DFC-8B25-4936-A067-4B1EE831FA2D )`

Remarks
— In the following function specifications, the name `ra` will be used to refer to the provider of this interface.

8.1.10.5.1.1 GetRange

Signature
```
uhErrorCode_t GetRange (
    [out] pInt32 pMinAmp,
    [out] pInt32 pMaxAmp );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the boundaries for the RF amplification level. A negative value indicates attenuation; a positive value indicates amplification. The value 0, always part of the range, indicates no amplification/attenuation.

Parameters

Table 706

Name	Description
`pMinAmp`	Output parameter returning the minimum amplification level.
`pMaxAmp`	Output parameter returning the maximum amplification level.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— None

Postcondition
— `*pMinAmp == ra.minAmp`
— `*pMaxAmp == ra.maxAmp`

8.1.10.5.1.2 Set

Signature
`uhErrorCode_t Set ( [in] Int32 amp );`

Qualifiers
— synchronous

— single-threaded

Description
This function sets the RF amplification level. A negative value indicates attenuation; a positive value indicates amplification. The value 0 indicates no amplification or attenuation.

Parameters

Table 707

Name	Description
`amp`	Input parameter indicating the new amplification level.

Return Values
Standard.

Precondition
`ra.minAmp ≤ amp ≤ ra.maxAmp`

Action
— Modify `ra.currentAmp`

Postcondition
`ra.currentAmp == amp`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.10.5.1.3 Get

Signature
`uhErrorCode_t Get ( [out] pInt32 pAmp );`

Qualifiers
— synchronous

— thread-safe

Description
This function returns the RF amplification level, as it was set via the Set function. A negative value indicates attenuation; a positive value indicates amplification. The value 0 indicates no amplification or attenuation.

Parameters

Table 708

Name	Description
`pAmp`	Output parameter returning the set amplification level.

Return Values
Standard.

Precondition
`True`

Action
— None

Postcondition
`*pAmp == ra.currentAmp`

8.1.11 UhlSigStrength

8.1.11.1 Concepts

8.1.11.1.1 Introduction

The function of a `SigStrength` is to measure the strength of an IF signal. The strength is measured in logical units. The higher the value the stronger is the signal. The relationship between logical unit and physical value is platform-specific and does not covered in this interface.

Measured strength can be used for sorting of signals or for distinction between signal and noise.

The measurement of signal strength can influence the signal coming out of component. The signal becomes stable again as soon as measurement is completed or abort of measurement proceed.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.11.2 Types & Constants

8.1.11.2.1 Public Types & Constants

8.1.11.2.1.1 Error Codes

None.

8.1.11.2.1.2 uhSigStrength_Notification_t

Signature

```
typedef enum _uhSigStrength_Notification_t {
    uhSigStrength_SubscriptionChanged = 0x00000001,
    uhSigStrength_MeasReady           = 0x00000002,
    uhSigStrength_MeasAborted         = 0x00000004
} uhSigStrength_Notification_t, *puhSigStrength_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the notification functions in the interface `uhISigStrengthNtf`. They can be used to individually turn on and off notifications.

Values

Table 709

Name	Description
`uhSigStrength_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhSigStrength_MeasReady`	Represents `OnMeasReady`.
`uhSigStrength_MeasAborted`	Represents `OnMeasAborted`.

8.1.11.2.1.3 uhSigStrength_Notifications_t

Signature

```
typedef Int32 uhSigStrength_Notifications_t, *puhSigStrength_Notifications_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for `uhSigStrength_Notification_t`.

8.1.11.2.1.4 uhSigStrength_AllNotifications

Signature

```
const uhSigStrength_Notifications_t uhSigStrength_AllNotifications = 0x00000007;
```

Qualifiers

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The set of all notifications of `uhSigStrength_Notification_t`.

8.1.11.2.1.5 uhSigStrength_ValStat_t

Signature
```
typedef enum _uhSigStrength_ValStat_t {
    uhSigStrength_ValStatInvalid = -2,
    uhSigStrength_ValStatUnknown = -1,
    uhSigStrength_ValStatKnown   =  1
} uhSigStrength_ValStat_t, *puhSigStrength_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 710

Name	Description
`uhSigStrength_ValStatInvalid`	The signal strength property is temporarily not available.
`uhSigStrength_ValStatUnknown`	The signal strength property is available, yet could not be determined.
`uhSigStrength_ValStatKnown`	The signal strength property is available and determined.

8.1.11.3 Logical Component

8.1.11.3.1 Interface-Role Model

Figure 74 depicts the interface-role model of the `SignalStrength` logical component. It shows the control and notification interfaces and the roles involved with this logical component.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

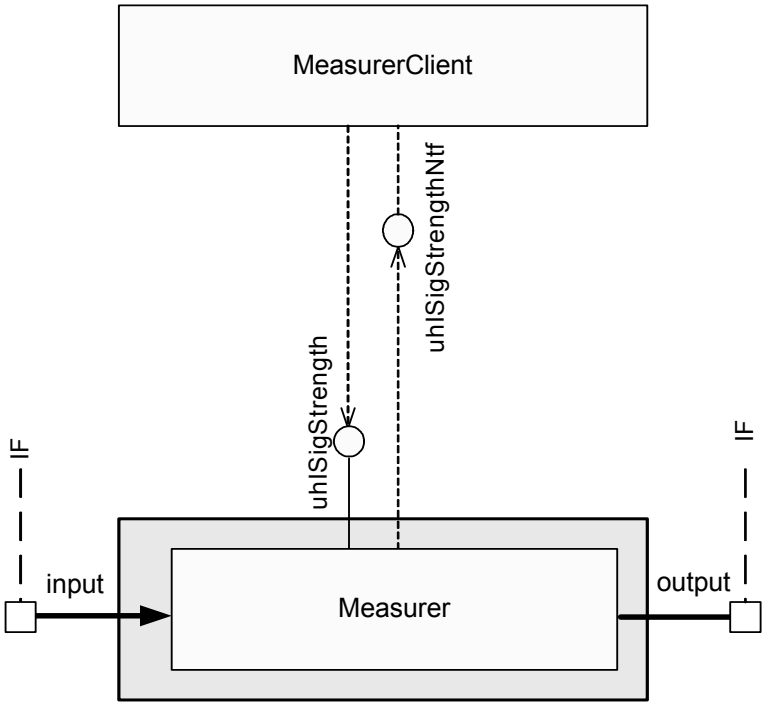


Figure 74 — Interface-Role Model

The grey box in Figure 74 denotes a `SigStrength` logical component containing the Measurer role providing one mandatory (solid line) control interface to the clients.

Clients can optionally provide a notification interface allowing them to receive notifications of events in the `Measurer`.

8.1.11.3.2 Diversity

8.1.11.3.2.1 Provided Interfaces

Table 711

Role	Interface	Presence
`Measurer`	`uhISigStrength`	mandatory
`MeasurerClient`	`uhISigStrengthNtf`	optional

8.1.11.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of the logical component. For a description of the attributes see 7.2.1.4 Roles, and for how they are initialized see subclause 8.1.11.3.3.

Table 712

Role	Attribute
`Measurer`	`minSigStrength`
`Measurer`	`maxSigStrength`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.11.3.2.3 Constraints

See the Invariants of the `Measurer` (subclause 8.1.11.4.1) role.

8.1.11.3.3 Instantiation

8.1.11.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 713

Type	Object	Multiplicity
`Measurer`	`meas`	1

8.1.11.3.3.2 Initial State

```
meas. currentSigStrengthStatus == uhSigStrength_ValStatUnknown

meas.currentSigStrength == meas.minSigStrength

meas.measOngoing == False
```

8.1.11.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.1.11.4 Roles

8.1.11.4.1 Measurer

Signature

```
role Measurer {
    const UInt32            minSigStrength;
    const UInt32            maxSigStrength;
    uhSigStrength_ValStat_t currentSigStrengthStatus;
    UInt32                  currentSigStrength;
    Bool                    measOngoing;
}
```

Qualifiers

— root

Description

A `Measurer` controls the signal strength attributes.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 714

Name	Description
`minSigStrength`	The minimum signal strength.
`maxSigStrength`	The maximum signal strength.
`currentSigStrengthStatus`	The validity status of the signal strength.
`currentSigStrength`	The current value of the signal strength.
`measOngoing`	Indicates if a measurement is ongoing (`True`) or not (`False`).

Invariants

```
minSigStrength <= currentSigStrength <= maxSigStrength
```

Instantiation

A `Measurer` role is always instantiated as part of a `SigStrength` logical component; see subclause 8.1.11.3.3.

Streaming Behavior

The `Measurer` role can distort output signal by doing measurement.

Active Behavior

The `Measurer` has standard notification behavior (see `uhISigStrengthNtf`).

8.1.11.4.2 MeasurerClient

Signature

```
role MeasurerClient {}
```

Qualifiers

— actor

Description

A `MeasurerClient` role represents a user of the control interfaces offered by the `Measurer` role. It can optionally provide the notification interfaces to receive notifications of events generated by the `Measurer` role. No constraints apply to the behavior of the `MeasurerClient` role other than those implied by Figure 74.

8.1.11.5 Interfaces

8.1.11.5.1 uhISigStrength

Qualifiers

None.

Description

This interface of a `Measurer` role provides control over signal strength measurement process.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 1ACB1CF4-BE6F-4645-83BA-849DC485571D )`

Remarks
— In the following function specifications the name `meas` will be used to refer to the provider of this interface.

8.1.11.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhISigStrengthNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhSigStrength_Notifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.11.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhISigStrengthNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhSigStrength_Notifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.11.5.1.3 GetSigStrengthRange

Signature
```
uhErrorCode_t GetSigStrengthRange (
    [out] pUInt32 pMinStrength,
    [out] pUInt32 pMaxStrength );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and maximum signal strength. The value of the minimum and maximum signal strength denotes a logical number, where the minimum represents the weakest and the maximum represents the strongest signal strength.

Parameters

Table 715

Name	Description
`pMinStrength`	Output parameter returning the minimum signal strength.
`pMaxStrength`	Output parameter returning the maximum signal strength.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMinStrength == meas.minSigStrength`
— `*pMaxStrength == meas.maxSigStrength`

8.1.11.5.1.4 Measure

Signature
`uhErrorCode_t Measure ( );`

Qualifiers
— asynchronous

— single-threaded

Description
Starts the measurement of the signal strength. When the measurement is ready, this is notified to the client. When this function is called again before the previous measurement is finished, the second call has no effect. If signal is too weak to be measured, then `meas.minSigStrength` is returned. If signal is too strong to be measured, then `meas.maxSigStrength` is returned.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Action
— If `(meas.measOngoing == False)`
  — Modify `meas.measOngoing`

Postcondition
— `meas.currentSigStrengthStatus == uhSigStrength_ValStatUnknown`
— `meas.measOngoing == True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Asynchronous Action
— Modify `meas.measOngoing`
— Modify `meas.currentSigStrengthStatus`
— Modify `meas.currentSigStrength`
— Notify `OnMeasReady`

Asynchronous Postcondition
— `meas.measOngoing == False`
— `meas.currentSigStrengthStatus != uhSigStrength_ValStatUnknown`
— `meas.currentSigStrength ==` newly measured signal strength `or meas.minSigStrength or meas.maxSigStrength`

8.1.11.5.1.5 Abort

Signature
```
uhErrorCode_t Abort ( );
```

Qualifiers
— asynchronous

— single-threaded

Description
Aborts an ongoing signal strength measurement.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Action
— If `(meas.measOngoing == True)`
  — stop the measurement

Postcondition
`True`

Asynchronous **Action**
— Modify `meas.measOngoing`
— Modify `meas.currentSigStrengthStatus`
— Modify `meas.currentSigStrength`
— Notify `OnMeasAborted`

Asynchronous Postcondition
— `meas.measOngoing == False`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— meas.currentSigStrengthStatus != uhSigStrength_ValStatKnown
— meas.currentSigStrength == meas.minSigStrength

8.1.11.5.1.6 GetSigStrength

Signature

```
uhErrorCode_t GetSigStrength (
    [out] pBool pMeasOngoing,
    [out] puhSigStrength_ValStat_t pStatus,
    [out] pUInt32 pSigStrength );
```

Qualifiers

— synchronous

— thread-safe

Description

Function returning the current signal strength including its validity status and if the measurement is ongoing or not. For example, when a measurement is ongoing the value of returned signal strength will not be reliable. The returned signal strength is a logical value that lies between the minimum and maximum signal strength. The higher the value of the returned signal strength the stronger the incoming signal is.

Parameters

Table 716

Name	Description
pMeasOngoing	Output parameter indicating if measurement is still ongoing (True) or not (False).
pStatus	Output parameter returning the validity status of the signal strength.
pSigStrength	Output parameter returning the current signal strength.

Return Values

Standard.

Precondition

True

Action

None.

Postcondition

— *pMeasOngoing == meas.measOngoing
— *pStatus == meas.currentSigStrengthStatus
— *pSigStrength == meas.currentSigStrength

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.11.5.2 uhISigStrengthNtf

Qualifiers
— callback

Description
This interface can be provided by an instance of the `MeasurerClient` role that wants to receive notifications of events that occur in the `Measurer`. The `Subscribe` function of the `uhISigStrength` interface can be used to subscribe to these events.

Interface ID
`uuid( 9FA9E45D-81C1-4da8-AB42-938AAE39462D )`

8.1.11.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhSigStrength_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.11.5.2.2 OnMeasReady

Signature
```
[async] Void OnMeasReady (
    [in] UInt32 cookie,
    [in] uhSigStrength_ValStat_t status,
    [in] UInt32 sigStrength );
```

Qualifiers
None.

Description
Notifies that a measurement is ready, and provides the measured signal strength including its validity status.

Parameters

Table 717

Name	Description
`cookie`	The value that was passed by an instance of the `MeasurerClient` role at subscription time.
`status`	The validity status of the notified signal strength.
`sigStrength`	The signal strength.

Return Values
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.11.5.2.3 OnMeasAborted

Signature
```
[async] Void OnMeasAborted ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
Notifies that the ongoing measurement is aborted.

Parameters

Table 718

Name	Description
`cookie`	The value that was passed by an instance of the `MeasurerClient` role at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.12 hlStcDec

8.1.12.1 Concepts

8.1.12.1.1 Introduction

This document describes the API of the STC Decoder logical component. A STC Decoder is associated with a de-multiplexer present in the system.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The STC decoder can operate in one of two modes: push or pull. In both modes, the STC decoder maintains the STC clock.

In push mode, the frequency and actual values of STC clock are locked to the incoming PCR values. The push mode is used for instance when the transport stream is received by a tuner.

In pull mode, the frequency is determined by the platform[16], and the actual values are linked to the PTS values of either the video or audio elementary stream. The pull mode is for instance often used when the client injects a Single Program Transport Stream (which it for example read from a hard disk) into the platform for playback.

Whether the STC decoder operates in push or pull mode depends on the context in which it is used (it should be evident from the context whether the STC decoder will operate in push or pull mode). The STC decoder does not provide control to enable push or pull mode.

The STC decoder also provides functions to set alarms.

8.1.12.1.2 STC Decoder

A digital MPEG stream contains different components that are decoded separately. Typical components are:

— video

— audio

— DVB subtitling

— closed captioning

— Teletext

— other private data streams

In general, presentation of this data should be synchronized. To achieve this, MPEG PES packets contain time references. In general, two time references are used:

a) Decoding Time Stamp (DTS), which defines the moment that decoding of the encoded data of the PES packet should be started.

b) Presentation Time Stamp (PTS), which defines the moment that the decoded data of the PES packet should be presented.

The time base is equal for all components that make up a program. Data containing the same PTS value must be presented at the same time. Therefore, a System Time Clock (STC) is associated with the MPEG stream, which provides this time base.

In push mode, the STC is a 27 MHz clock that has to be recovered from the Program Clock References (PCR) that, at a regular basis, have been put in the stream by the network operator (where a PCR is just a sample of a 27 MHz clock), see Figure 75. The STC Decoder (STCD) maintains the STC clock, the frequency and actual values of which are locked to the incoming PCR values[17]. Though the STC is a 27 MHz clock, time values that are used in the control interface of STCD are 90 kHz values, 33-bit wide, as a resolution of 27 MHz is too fine grained to communicate to a middleware stack (see also Figure 75).

---

16 Although the platform determines the frequency of the STC clock, the frequency will be (within certain bounds) 27MHz.

[17]The frequency of PCR is not exactly 27MHz. It may vary within certain bounds. Therefore, the actual speed of the STC is derived from the PCRs

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

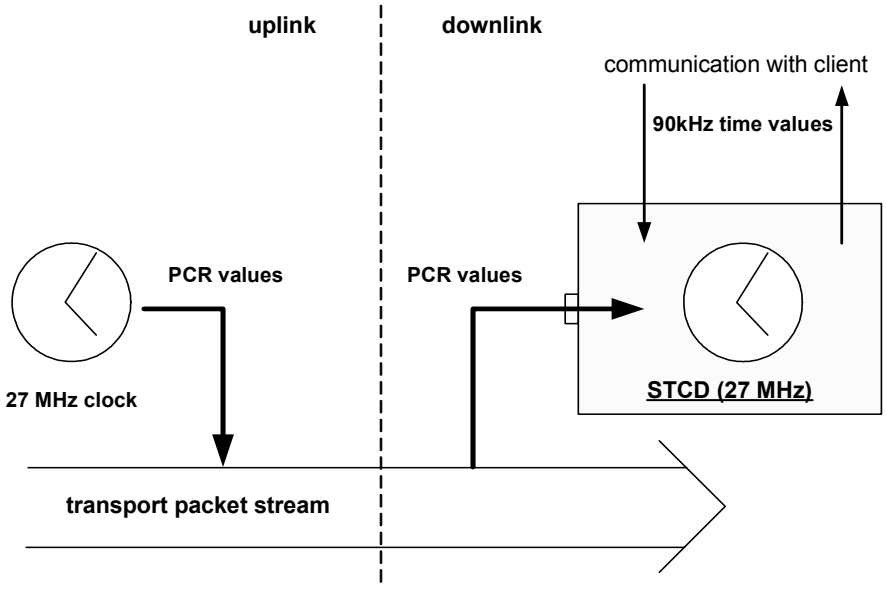


Figure 75 — General model of STC synchronization (in case of push mode).

In pull mode, the STC is a 27MHz autonomously running clock (when playing at normal speed) whose values in time are linked to the PTS values of the video or audio elementary stream (i.e., when a frame with a particular PTS is rendered, the STC value will match the PTS[18]). The STC decoder ignores the PCR values in pull mode.

In pull mode, certain trick modes are possible. Which trick modes are possible depends upon the context in which the STC decoder is used. For instance, the ATSC decoder 7.1.2, which uses this STC decoder specification, offers a number of trick modes. Table 719 shows the relation between the trick modes and the STC decoder.

Table 719 — Relation between trick modes and the STC decoder.

Trick mode	Relation with STC decoder
Playback with a different speed (either slower or faster than normal playback)	The STC clock is slowed down or sped up accordingly. For instance, when playing back at ½ speed, the STC runs at ½ speed (i.e., 13.5MHz).
Pause	The STC clock is paused as well (on the presentation time of the paused frame).
Decoding of one I-frame at a time.	The STC clock is not maintained (its value is invalid), as maintaining it does not make sense (STC is not used to determine when to present a frame).
Decoding of only the I-frames in a stream, where the frames are rendered without regard to the PTS values (but as soon as possible) and where the input stream may leave out frames that do not need to be decoded.	The STC clock is not maintained (its value is invalid), as maintaining it does not make sense (STC is not used to determine when to present a frame).
Decoding of only the I- and P-frames in a stream, where the frames are rendered without regard to the PTS values (but as soon as possible) and where the input stream may leave out frames that do not need to be decoded.	The STC clock is not maintained (its value is invalid), as maintaining it does not make sense (STC is not used to determine when to present a frame).

18 In case of a unannounced discontinuity this may be different, as explained in subclause 8.1.12.1.3.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.1.3 Irregularities

Normally, the difference between subsequently received PCR or PTS values corresponds to the elapsed time between their respective receptions. Sometimes, this may not be the case. If the deviation of a received PCR or PTS value from the expected value is not within certain bounds, that PCR or PTS is regarded as the start of a new time base. This is called a *discontinuity*. We distinguish two kinds of discontinuity: announced and unannounced discontinuities. The former are intended discontinuities; there is an indication in the stream that a discontinuity is about to occur. The latter are not predicted and could be the result of some error. In case of an announced discontinuity, the STC will "jump" to the new PCR or PTS value. Its frequency remains the same until the next PCR or PTS is received and a reliable derivation of the actual frequency can be made again. If there is an unannounced discontinuity, the STC becomes *unlocked*, meaning that the coupling of STC and PCRs or PTSs is lost. The STC clock becomes free running in that state. When a number of PCRs or PTSs have been received that are within bounds, the STC will become *locked* again. See Figure 76 (In this example, the number of well-behaved PCRs or PTSs that have to be received to become locked again is 3). Another way of becoming unlocked is when PCRs or PTSs are late. Within 100 ms of the reception of the last PCR or within 700 ms of the reception of the last PTS, a new one must arrive. As with unannounced discontinuities the STC becomes locked again when a number of well-behaved PCRs or PTSs have been received.

NOTE The STC can make jumps that are the start of a new time base when an announced discontinuity occurs or when it becomes locked after having been unlocked for a while.

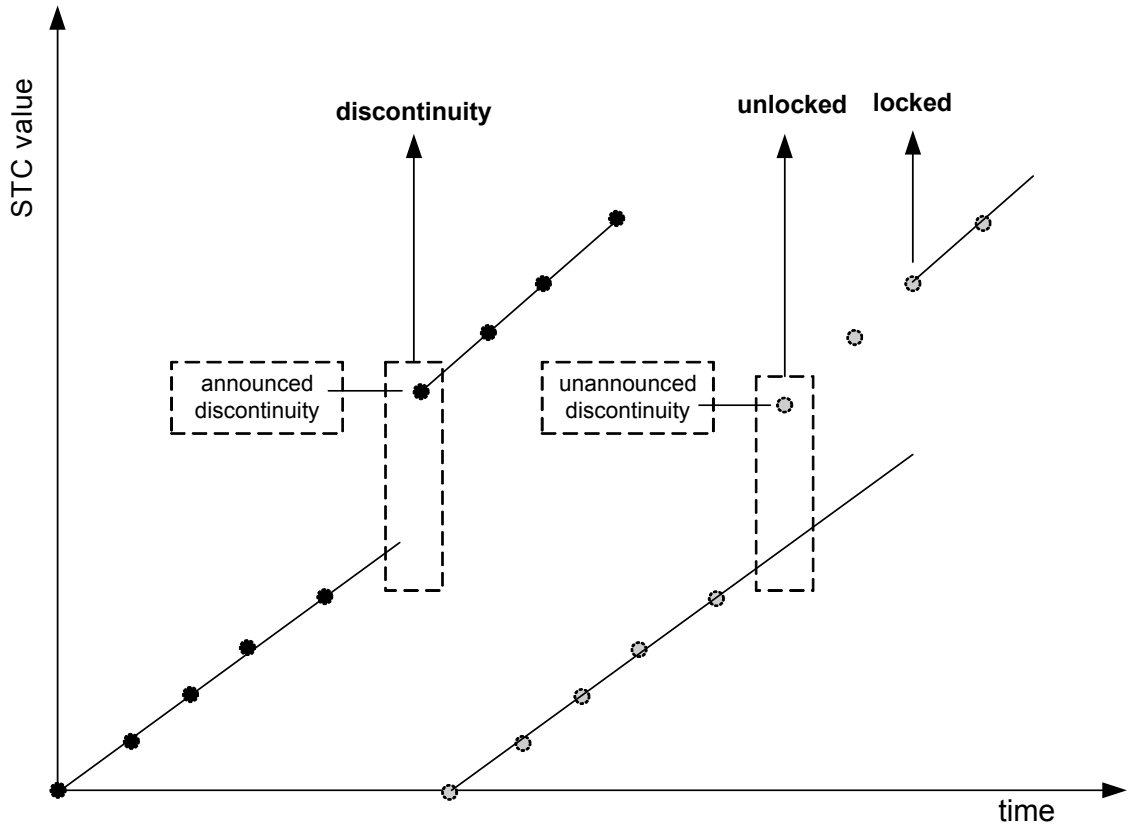


Figure 76 — STC behavior in case of discontinuities.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.2 Types & Constants

8.1.12.2.1 Public Types & Constants

8.1.12.2.1.1 Error Codes

Signature

```
const uhErrorCode_t UH_ERR_STCDEC_ALARM_NOT_AVAILABLE = 0x00000001;
const uhErrorCode_t UH_ERR_STCDEC_ALARM_IN_PAST       = 0x00000002;
const uhErrorCode_t UH_ERR_STCDEC_ALARM_EXPIRED       = 0x00000003;
const uhErrorCode_t UH_ERR_STCDEC_NOT_LOCKED          = 0x00000004;
```

Qualifiers

— error-codes

Description

Defines all the non-standard error codes, returned by the functions of the interface(s) of a STC decoder.

Constants

Table 720

Name	Description
UH_ERR_STCDEC_ALARM_NOT_AVAILABLE	Alarm is not available.
UH_ERR_STCDEC_ALARM_IN_PAST	The alarm time is in the past.
UH_ERR_STCDEC_ALARM_EXPIRED	The alarm has expired.
UH_ERR_STCDEC_NOT_LOCKED	The STC decoder is not locked

8.1.12.2.1.2 uhStcDec_Notification_t

Signature

```
typedef enum _uhStcDec_Notification_t {
    uhStcDec_SubscriptionChanged = 0x00000001,
    uhStcDec_Locked              = 0x00000002,
    uhStcDec_Unlocked            = 0x00000004,
    uhStcDec_Discontinuity       = 0x00000008,
    uhStcDec_Invalid             = 0x00000010
} uhStcDec_Notification_t, *puhStcDec_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represents notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIStcDecNtf`. These values can be used to individually turn on and off notifications.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 721

Name	Description
uhStcDec_SubscriptionChanged	Represents OnSubscriptionChanged.
uhStcDec_Locked	Represents OnLocked
uhStcDec_Unlocked	Represents OnUnlocked.
uhStcDec_Discontinuity	Represents OnDiscontinuity
uhStcDec_Invalid	Represents OnInvalid

8.1.12.2.1.3 uhStcDec_Notifications_t

Signature

```
typedef Int32 uhStcDec_Notifications_t, *puhStcDec_Notifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represents sets of notification functions which are typically constructed by logical OR-ing of values of type uhStcDec_Notification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.12.2.1.4 uhStcDec_AllNotifications

Signature

```
const uhStcDec_Notifications_t uhStcDec_AllNotifications = 0x0000001F;
```

Qualifiers

none.

Description

Defines the set of all values of type uhStcDec_Notification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.1.12.2.1.5 uhStcDec_AlarmNotification_t

Signature

```
typedef enum _uhStcDec_AlarmNotification_t {
    uhStcDec_AlarmSubscriptionChanged = 0x00000001,
    uhStcDec_AlarmExpired             = 0x00000002,
    uhStcDec_AlarmCancelled           = 0x00000004
} uhStcDec_AlarmNotification_t, *puhStcDec_AlarmNotification_t;
```

Qualifiers

— enum-element

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represents notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIStcDecAlarmNtf`. These values can be used to individually turn on and off notifications..

Values

Table 722

Name	Description
`uhStcDec_AlarmSubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhStcDec_AlarmExpired`	Represents `OnExpired`.
`uhStcDec_AlarmCancelled`	Represents `OnCancelled`.

8.1.12.2.1.6 uhStcDec_AlarmNotifications_t

Signature
```
typedef Int32 uhStcDec_AlarmNotifications_t, *puhStcDec_AlarmNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represents sets of notification functions which are typically constructed by logical OR-ing of values of type uh`StcDec_AlarmNotification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.1.12.2.1.7 uhStcDec_AlarmAllNotifications

Signature
```
const uhStcDec_AlarmNotifications_t uhStcDec_AlarmAllNotifications = 0x00000007;
```

Qualifiers
none.

Description
Defines the set of all values of type uh`StcDec_AlarmNotification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.2.1.8 uhStcDec_ValStat_t

Signature

```
typedef enum _uhStcDec_ValStat_t {
    uhStcDec_ValStatInvalid = -2,
    uhStcDec_ValStatKnown   =  1
} uhStcDec_ValStat_t, *puhStcDec_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 723

Name	Description
uhStcDec_ValStatInvalid	The property/properties are temporarily not available.
uhStcDec_ValStatKnown	The property/properties are available and determined.

8.1.12.2.2 Model Types & Constants

8.1.12.2.2.1 AlarmState_t

Signature

```
typedef enum {
    AlarmInActive,
    AlarmActive,
    AlarmCancelRequested
} AlarmState_t, *pAlarmState_t;
```

Qualifiers
None.

Description
The `AlarmState_t` type represents the state of the alarm.

Table 724

Name	Description
AlarmInActive	Alarm has been created, but not active.
AlarmActive	Alarm is currently active.
AlarmCancelRequested	A cancel has been requested on the alarm.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.2.2.2 LockStatus_t

Signature

```
typedef enum {
    Invalid,
    Unlocked,
    Locked
} LockStatus_t, *pLockStatus_t;
```

Qualifiers
None.

Description
This type represents the lock status of the STC.

Values

Table 725

Name	Description
`Invalid`	The lock status of the STC is temporarily not available.
`Unlocked`	The STC is unlocked.
`Locked`	The STC is locked.

8.1.12.3 Logical Component

8.1.12.3.1 Interface-Role Model

Figure 77 defines the interface-role model of STCD. It shows the interfaces, role and input used in the specification of STCD.

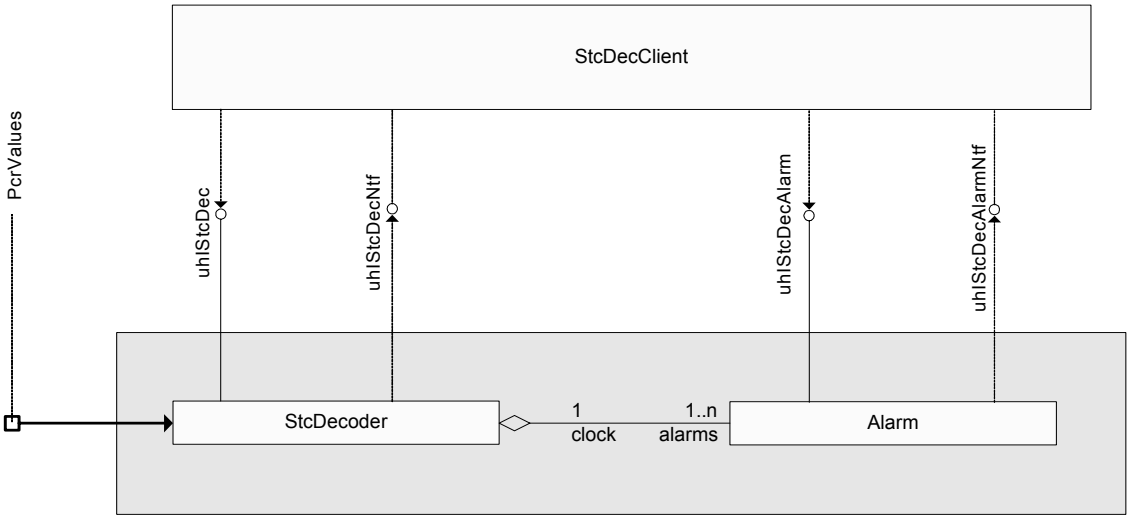


Figure 77 — Interface-Role Model of STC Decoder

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.3.2 Diversity

8.1.12.3.2.1 Provided Interfaces

Table 726

Role	Interface	Presence
StcDecoder	uhIStcDec	Mandatory
Alarm	uhIStcDecAlarm	Mandatory
StcDecClient	uhIStcDecNtf	Optional
	uhIStcDecAlarmNtf	Optional

8.1.12.3.2.2 Configurable Items

Table 727

Role	Attribute
StcDecoder	maxNrAlarms

8.1.12.3.2.3 Constraints

— maxNrAlarms ≥ 0

8.1.12.3.3 Instantiation

8.1.12.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 728

Type	Object	Multiplicity
StcDec	stcDecoder	1
StcDecAlarm	alarm[]	stcDecoder.maxNrAlarms

8.1.12.3.3.2 Initial State

The STC Decoder is created by the Demux instance.

The following constraints apply to the initial state of a logical component instance:

— stcDecoder.nrAlarms == 0
— stcDecoder.lockStatus == Invalid
— stcDecoder.discontinuityStatus == uhStcDec_ValStatInvalid
— stcDecoder.currentValue == <platform specific>
— stcDecoder.currentValueStatus == uhStcDec_ValStatInvalid
— For each i in {0 .. stcDecoder.maxNrAlarms-1}
  — alarm[i].alarmState == AlarmInActive.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.1.12.4 Roles

8.1.12.4.1 StcDecoder

Signature

```
role StcDecoder {
    const UInt32        maxNrAlarms;
    UInt32              nrAlarms;
    setof(Alarm)        alarm;
    uhStcDec_ValStat_t  currentValueStatus;
    uhTimeStamp_t       currentValue;
    LockStatus_t        lockStatus;
    uhStcDec_ValStat_t  discontinuityStatus;
}
```

Qualifiers

— root

Description

An STC Decoder provides the following functionality:

— Synchronization of the STC with the incoming PCR or PTS values.

— Facilities for monitoring the STC.

— Detection and notification of irregularities in the incoming PCR or PTS values.

Independent Attributes

Table 729

Name	Description
maxNrAlarms	The maximum number of supported alarms.
alarm	The set of all alarms.
currentValueStatus	The validity status of the STC value.
currentValue	The current value of the STC. Maintains a 33 bit time counter running at ~90KHz frequency based on incoming PCRs or PTSs.
lockStatus	The lock status of the STC.
discontinuityStatus	The validity status of the STC discontinuity status.

Dependent Attributes

Table 730

Name	Description
nrAlarms	The number of allocated alarms.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Invariants
`nrAlarms <= maxNrAlarms`

Streaming Behavior
The STCD receives PCR or PTS values, with which it recovers the STC (currentValue). With each incoming PCR or PTS, the desired frequency for the STC is calculated. Ideally, PCR_t or PTS_t ==STC_t should hold, where PCR_t is the PCR received at time t, PTS_t is the PTS received at time t and STC_t is the value of the STC at time t. It is guaranteed that | PCR_t or PTS_t - STC_t | <= δ for some δ. In case of an announced discontinuity (| PCR_t or PTS_t - STC_t | > δ), the STC jumps to PCR_t or PCR_t. When an unannounced discontinuity is detected or 100ms have passed since the reception of the last PCR or 700 ms since the reception of the last PTS[19], the STC becomes unlocked (`lockStatus == Unlocked`). In that state, no guarantees are made on the value of the STC. When sufficient (sufficient is platform instance specific) PCRs or PTSs that arrived in time and did not show any unannounced discontinuities have been received, the STC becomes locked again (`lockStatus == Locked`).

Active Behavior
The StcDecoder has standard notification behavior; see `uhIStcDecNtf` for the events that can be notified

8.1.12.4.2 StcDecClient

Signature
```
role StcDecClient {}
```

Qualifiers
— actor

Description
A STC Decoder Client represents a user of the `uhIStcDec` and `uhIStcDecAlarm` interface.

8.1.12.4.3 Alarm

Signature
```
role Alarm {
    uhTimeStamp_t  minExpireTime;
    uhTimeStamp_t  maxExpireTime;
    AlarmState_t   alarmState;
}
```

Qualifiers
— None.

Description
The client can set a STC alarm time window and by this activating an alarm. When the STC is in this time window, the client will be notified. The alarm can still be cancelled until the alarm expired notification occurs. When the STC becomes unlocked, the alarms are still active and expire based on a free running clock. If the client wants, he may cancel the alarms on receiving the unlocked notification.

---

19 The 700ms time period is adapted according to the actual playback speed. For example, when playing at half speed, the 700ms becomes 1400ms.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 731

Name	Description
`minExpireTime`	The minimum value of STC when the alarm should expire
`expireTime`	The maximum value of STC when the alarm should expire.
`alarmState`	The state the alarm is in.

Instantiation
An alarm is created when the client makes a successful call to `AllocateAlarm`.

Streaming Behavior
None.

Active Behavior
The Alarm has standard notification behavior; see `uhIStcDecAlarmNtf` for the events that can be notified

Remarks on Cancellation

As the cancellation of alarms is asynchronous, the client registers for the cancellation of the alarm by calling `CancelAlarm` of interface `uhIStcDecAlarm`. The actual cancellation will be notified by an OnCancelled notification by the STCD

Noteworthy transitions are:

— If the alarm expires in the `AlarmCancelRequested` state, the client will be notified of the cancellation and not of the expiration.

— If `CancelAlarm`() is called in state `AlarmCancelRequested` leaves the state unchanged and does not produce an error message.

Remarks on Expiration

Due to discontinuities, the STC does not behave like an ideal continuous clock. It can become unlocked and locked repeatedly, and can make jumps (back and forth). This has an impact on the behavior of alarms. In Figure 78 an erratic STC is shown and the effect of its behavior on three alarms: A,B and C. Alarm A expires at the “expected“ time. Alarm B expires earlier and Alarm C expires much later than would be expected. Theoretically an alarm could never expire, because the STC keeps jumping back before it reaches the expire time. Via the `OnDiscontinuity` and `OnLocked` events the client can be warned of erratic STC behavior and can chose to cancel the alarms.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

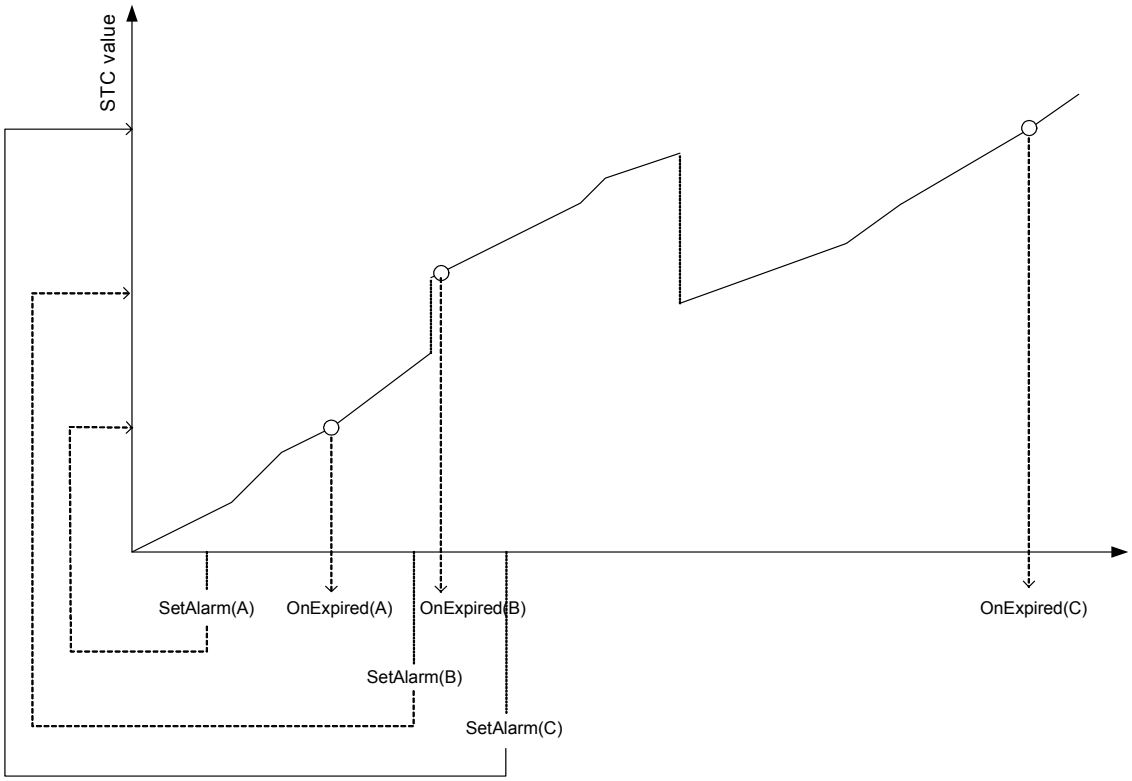


Figure 78 — Behavior of alarms in the presence of discontinuities.

8.1.12.5 Interfaces

8.1.12.5.1 uhlStcDec

Qualifiers
None

Description
This interface of an STCD provides means to monitor the STC.

Interface ID
`uuid( 3CFC6780-85DC-4d60-AF8F-6C55FC298E91 )`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIStcDecNtf            *pINotify,
    [in] UInt32                   cookie,
    [in] uhStcDec_Notifications_t notifs
);
```

Qualifiers

— subscribe-function

8.1.12.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIStcDecNtf            *pINotify,
    [in] UInt32                   cookie,
    [in] uhStcDec_Notifications_t notifs
);
```

Qualifiers

unsubscribe-function

8.1.12.5.1.3 GetTime

Signature

```
uhErrorCode_t GetTime (
    [out] puhStcDec_ValStat_t pStatus,
    [out] puhTimeStamp_t      pValue
);
```

Qualifiers

— synchronous

— threaded-safe

Description

The current value of the STC is returned. The clock is a 33 bit value running at 90KHz, based on the incoming PCRs or PTSs.

Parameters

Table 732

Name	Description
pStatus	Output parameter returning the validity status of the clock.
pValue	Output parameter returning the current value of the clock.

Return Values

Only non-standard return values are listed

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 733

Value	Description
`uh_ERR_STCDEC_NOT_LOCKED`	The STC is not locked.

Precondition
`True`

Action
— None

Postcondition
— *pStatus `== stcDecoder.currentValueStatus`

— **pValue* `== stcDecoder.currentValue`

Remarks

— If STC decoder is not locked, then the current free running value will be returned. The return value just provides a warning that this clock value may be unreliable.

8.1.12.5.1.4 GetMaxNrAlarms

Signature
```
uhErrorCode_t GetMaxNrAlarms (
    [out] pUInt32 pMaxNrAlarms
);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum number of alarms that can be allocated.

Parameters

Table 734

Name	Description
`pMaxNrAlarms`	Output parameter returning the maximum number of alarms.

Return Values
Standard

Precondition
`True`

Action
— None

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— *pMaxNrAlarms `==` `stcDecoder.maxNrAlarms`

8.1.12.5.1.5 GetAvailableNrAlarms

Signature
```
uhErrorCode_t GetAvailableNrAlarms (
    [out] pUInt32 pAvailableNrAlarms
);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the available number of alarms that can be allocated.

Parameters

Table 735

Name	Description
`pAvailableNrAlarms`	Output parameter returning the available number of alarms.

Return Values
Standard

Precondition
`True`

Action
— None

Postcondition
— *pAvailableNrAlarms* `==` `stcDecoder.maxNrAlarms - stcDecoder.nrAlarms.`

8.1.12.5.1.6 AllocateAlarm

Signature
```
uhErrorCode_t AllocateAlarm (
    [out] uhIStcDecAlarm **ppIAlarm
);
```

Qualifiers
— synchronous

— thread-safe

©ISO/IEC 2007 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Allocates one of the available alarms and returns a pointer to the `uhIStcDecAlarm` interface of the allocated alarm.

Parameters

Table 736

Name	Description
`ppIAlarm`	Output parameter returning a pointer to the interface `uhIStcDecAlarm` of the allocated alarm.

Return Values
Only non-standard return values are listed

Table 737

Value	Description
`uh_ERR_STCDEC_ALARM_NOT_AVAILABLE`	Maximum number of alarms reached.

Precondition
`True`

Action
— If `stcDecoder.nrAlarms < stcDecoder.maxNrAlarms`
  — Modify `stcDecoder.nrAlarms`

Postcondition
— If `stcDecoder.nrAlarms < stcDecoder.maxNrAlarms`
  — `*ppIAlarm` contains a pointer to the `uhIStcDecAlarm` interface of the allocated alarm.
  — alarm belongs to the set of alarms associated with the `stcDecoder.`
  — `stcDecoder.nrAlarms = stcDecoder.nrAlarms + 1.`
  — `alarm.alarmState = AlarmInactive.`

Remarks
The `StcDecoder.nrAlarms` shall be decremented when the Alarm interface is released.

8.1.12.5.2 uhIStcDecAlarm

Qualifiers
None

Description
This interface of an STC Decoder provides means to setting, cancelling and monitoring alarms (getting its current value, setting and cancelling alarms).

Interface ID
`uuid( 6838E328-C304-427c-A2F4-7077AE50A129 )`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.5.2.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIStcDecAlarmNtf            *pINotify,
    [in] UInt32                        cookie,
    [in] uhStcDec_AlarmNotifications_t notifs
);
```

Qualifiers

— subscribe-function

8.1.12.5.2.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIStcDecAlarmNtf            *pINotify,
    [in] UInt32                        cookie,
    [in] uhStcDec_AlarmNotifications_t notifs
);
```

Qualifiers

— unsubscribe-function

8.1.12.5.2.3 ActivateAlarm

Signature

```
uhErrorCode_t ActivateAlarm (
    [in] uhTimeStamp_t minValue,
    [in] uhTimeStamp_t maxValue,
    [in] Bool absolute
);
```

Qualifiers

— synchronous

— single-threaded

Description

Sets an alarm on a specified STC time window and activates it. If the specified minimum value has been reached or passed by the STC but is less than the maximum value, the function `OnExpired` on the corresponding notification interface is invoked by the Alarm.

The value of timestamp identifies a 33-bit value representing the STC in 90 kHz. Time stamps have a notion of past and future. The maximum value of the time stamp is 2^{33}-1. When the time stamp value indicates now, then the window

[now + 1, (now + max/2) mod (max + 1))

is in the future, and

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[(now + max/2) mod (max + 1), now)

is in the past (see Figure 79).

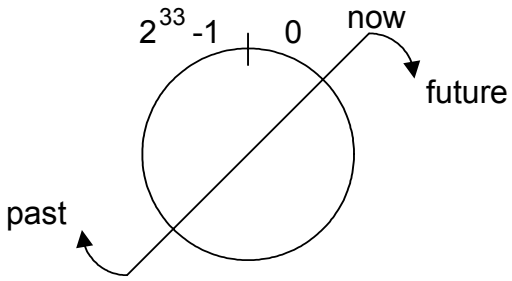


Figure 79 — The notion of past and future for time stamps.

Parameters

Table 738

Name	Description
`minValue`	The STC minimum value at which the alarm should expire.
`maxValue`	The STC maximum value at which the alarm should expire.
`absolute`	When True, the alarm is considered always to be in the future and the concept of past is ignored (resulting in an “extended future“).

Return Values

Only non standard return values are listed

Table 739

Value	Description
`uh_ERR_STCDEC_NOT_LOCKED`	The STC is not locked.

Precondition

— `Alarm.alarmState` in `{ AlarmInActive }`

Action

— `Alarm.minExpireTime =` MinValue
— `Alarm.maxExpireTime =` MaxValue

Postcondition

— `Alarm.expireTime ==` value
— `Alarm.alarmState ==AlarmActive`

Remarks

— The client must subscribe to the `OnExpired` notification.

— If the alarm is already active, ie., `alarm.alarmState == AlarmActive`, then the new value replaces the existing expireTime

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.12.5.2.4 CancelAlarm

Signature
```
uhErrorCode_t CancelAlarm (
);
```

Qualifiers
- asynchronous
- single-threaded

Description
Cancels an alarm set by the client.

Parameters
None

Return Values
Only non-standard return values are listed

Table 740

Value	Description
`UH_ERR_STCDEC_ALARM_EXPIRED`	Alarm has already expired.

Precondition
- `True`

Action
- If `alarm.alarmState == AlarmActive`
  - modify `alarm.alarmState`

Postcondition
- If `previous alarm.alarmState == AlarmActive`
  - `alarm.alarmState = AlarmCancelRequested`

Asynchronous Action
- If `alarm.alarmState == AlarmCancelRequested`
  - `alarm.alarmState = AlarmInActive`
  - Notify `OnCancel`led.

Asynchronous PostCondition
- `alarm.alarmState = AlarmInActive`

8.1.12.5.3 uhIStcDecNtf

Qualifiers
- callback

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface should be provided by each STC Decoder Client that wants to subscribe to notifications of events related to the STC decoder. The `Subscribe` function of `StcDecoder` can be used to subscribe to these events.

Interface ID
`uuid( 420276B5-0E71-4efd-82B6-447157EFA113 )`

8.1.12.5.3.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhStcDec_Notifications_t notifs
);
```

Qualifiers
— onsubscriptionchanged-function

8.1.12.5.3.2 OnLocked

Signature
```
[async] Void OnLocked (
    [in] UInt32 cookie
);
```

Qualifiers
None.

Description
This function is called when the STC has become locked.

Parameters

Table 741

Name	Description
`cookie`	The value that was passed by the STC Decoder Client at subscription time.

Return Values
None.

Precondition
— `True`

Action
— Any callback-compliant action.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— True

8.1.12.5.3.3 OnUnlocked

Signature
```
[async] Void OnUnlocked (
    [in] UInt32 cookie
);
```

Qualifiers
None.

Description
This function is called when the STC has become unlocked.

Parameters

Table 742

Name	Description
cookie	The value that was passed by the STC Decoder Client at subscription time.

Return Values
None.

Precondition
— True

Action
— Any callback-compliant action.

Postcondition
— True

8.1.12.5.3.4 OnInvalid

Signature
```
[async] Void OnInvalid (
    [in] UInt32 cookie
);
```

Qualifiers
None.

Description
This function is called when the lock status of the STC has become invalid.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 743

Name	Description
cookie	The value that was passed by the STC Decoder Client at subscription time.

Return Values
None.

Precondition
— True

Action
— Any callback-compliant action.

Postcondition
— True

8.1.12.5.3.5 OnDiscontinuity

Signature

```
[async] Void OnDiscontinuity (
    [in] UInt32 cookie,
    [in] uhStcDec_ValStat_t status
);
```

Qualifiers
None.

Description
This function is called when an earlier announced discontinuity occurred.

Parameters

Table 744

Name	Description
cookie	The value that was passed by the STC Decoder Client at subscription time.
status	The validity status of the notified discontinuity status.

Return Values
None.

Precondition
— True

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
— `True`

8.1.12.5.4 uhIStcDecAlarmNtf

Qualifiers
— callback

Description
This interface should be provided by each SDC that wants to subscribe to notifications of events related to the Alarm.

Interface ID
`uuid( EBC45552-C689-4208-845A-C1DCF0D1630C )`

8.1.12.5.4.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhStcDec_AlarmNotifications_t notifs
);
```

Qualifiers
— onsubscriptionchanged-function

8.1.12.5.4.2 OnExpired

Signature
```
[async] Void OnExpired (
    [in] UInt32 cookie,
    [in] Bool  lockState
);
```

Qualifiers
— callback

Description
This function is called when the alarm has expired.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 745

Name	Description
`cookie`	The value that was passed by the STC decoder Client at subscription time.
`lockState`	Current lock state of STC.

Return Values
None.

Precondition
— `True`

Action
— Any callback-compliant action.

Postcondition
— `True`

Remarks
Note that alarms can expire quite unexpectedly; see Figure 78.

8.1.12.5.4.3 OnCancelled

Signature

```
[async] Void OnCancelled (
    [in] UInt32 cookie
);
```

Qualifiers
— callback

Description
This function is called when the alarm has been cancelled.

Parameters

Table 746

Name	Description
`cookie`	The value that was passed by the STC decoder Client at subscription time.

Return Values
None.

Precondition
— `True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
— `True`

Remarks
None.

8.1.13 UhlTsDmx

8.1.13.1 Concepts

8.1.13.1.1 Introduction

This document describes the API of a demultiplexer for MPEG transport streams.

The function of a TS Demux is to filter specific content (e.g. PES packets, sections) from one transport stream (TS). The client can specify several criteria for filtering. The filtered data is routed to decoders or (client) middleware stacks. Several types of filters exist, each with their own filter criteria.

8.1.13.1.2 Demultiplexing

Stream and buffer

The TS Demux can have a number of Streams. A Stream models a stream of data. One buffer is being used per Stream for all data that is being filtered by the filters associated with this Stream. This buffer is not explicitly modelled. When the data is actually being routed to a client, the client has to manage the buffer explicitly.

TPH and PID Filter

A TPH Filter is a transport packet header filter that allows setting of filter criteria on some other fields than the PID. Figure 80 shows the structure of a transport packet header.

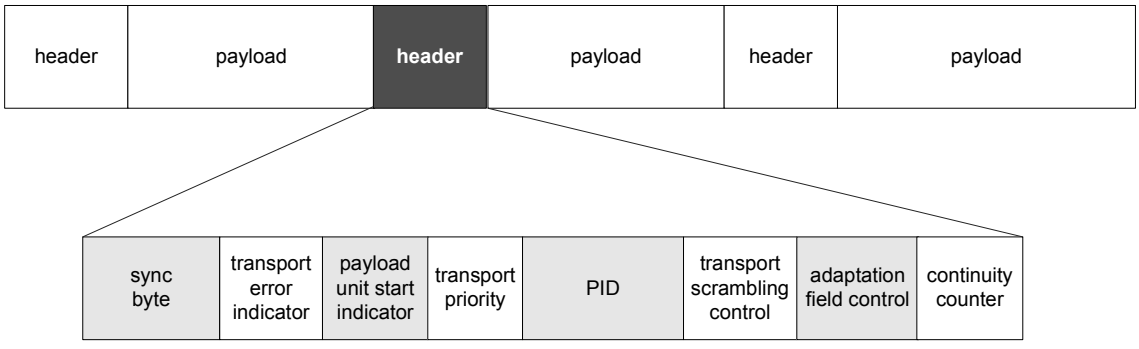


Figure 80 — Structure of a transport packet header

Typically, a Stream has one PID or TPH Filter. Only for remultiplexing of a single program, a Stream can have more of these filters. Each component of a program is conveyed on a separate PID. So, a PID or TPH Filter is needed for each component to be filtered.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section Filter, PES Filter, PES Notifier

Each PID Filter can have specialized filters attached to do additional filtering on the output of the PID Filters. These specialized filters are the following:

— PES Filters to filter PES packets,

— PES Notifiers to notify PES packets to clients,

— Section Filters for filtering of MPEG-2 sections.

TPH Filters can only have PES Filters and Notifiers attached. Each PID or TPH Filter can have at most one PES Notifier attached to notify PES packets to clients.

PID or TPH Filters can only have one type of specialized filters attached.

PCR Filter

The demultiplexer also has PCR Filters. A PCR Filter is used for filtering Program Clock Reference (PCR) data and routing it to a System Time Clock recovery mechanism. A PCR is a time stamp from which the timing of decoders is derived.

Descrambler

The descrambler and the demultiplexer are tightly bound and come in dedicated pairs (e.g. the TS demultiplexer and the DVB descrambler). That is why the interface of the Descrambler is also described in this document.

Content providers can protect their content through scrambling. The Descrambler descrambles streams at various levels (TS, PES, and section). Descrambling is only possible if the Descrambler has the right descrambling keys. If a customer is entitled to get access to the descrambled content, the client can get access to the proper descrambling keys which will then be fed to the Descrambler.

Figure 81 shows some TS Demux concepts.

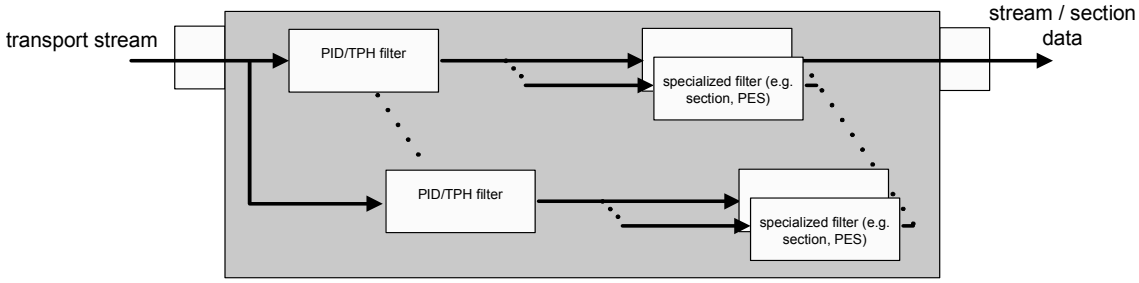


Figure 81 — Demultiplexer concepts

8.1.13.1.3 Typical scenario: playing a program

Figure 82 shows a typical scenario: playing a program. The thin arrows indicate associations among objects and the thick ones the direction of filtered data.

Streams ‘stream1', ‘stream2', and ‘stream3' extract respectively video, audio, and subtitles from the incoming transport stream and route their data to the respective decoders. Stream ‘stream4' filters Teletext data and routes it to memory after applying PES header filtering using a PES Filter. Streams ‘stream5' and ‘stream6' filter meta-data from the incoming transport stream that tells among other things on which PIDs the

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

components of the program are conveyed. Stream ‘stream7' filters ECMs from the incoming transport stream. For filtering EMMs another Stream with a Section Filter is needed. This is not shown in the picture as it would make it too crowded. This meta-data is packed in sections.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

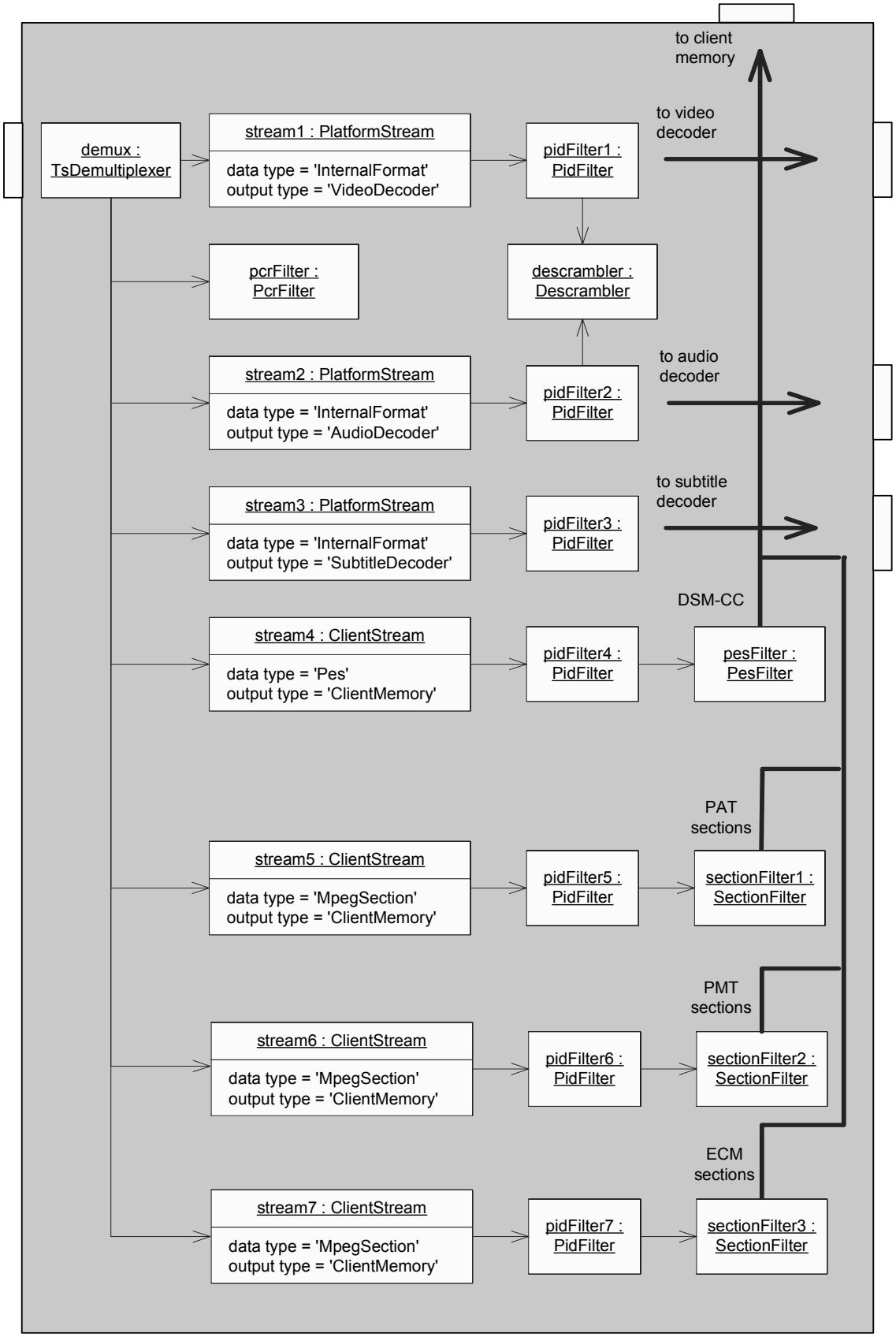


Figure 82 — Program play scenario

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.1.4 Single Program Transport Stream (SPTS) remultiplexing

Figure 83 depicts a SPTS generation scenario. This Single Program Transport Stream (SPTS) is routed to a remultiplexer.

Stream ‘stream1' is used for generating a SPTS and routing it to a remultiplexer. This SPTS is composed of one video component including PCR data, one audio component, one subtitle component, and DSM-CC data. Video and audio are descrambled before they are routed to the remultiplexer.

Furthermore, the two Streams ‘stream2' and ‘stream3' are for PAT and PMT section data retrieval respectively to client memory. The client can modify these sections and then re-insert them. Stream ‘stream4' filters ECMs from the incoming transport stream. For filtering EMMs another Stream with a Section Filter is needed. This is not shown in the picture as it would make it too crowded.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

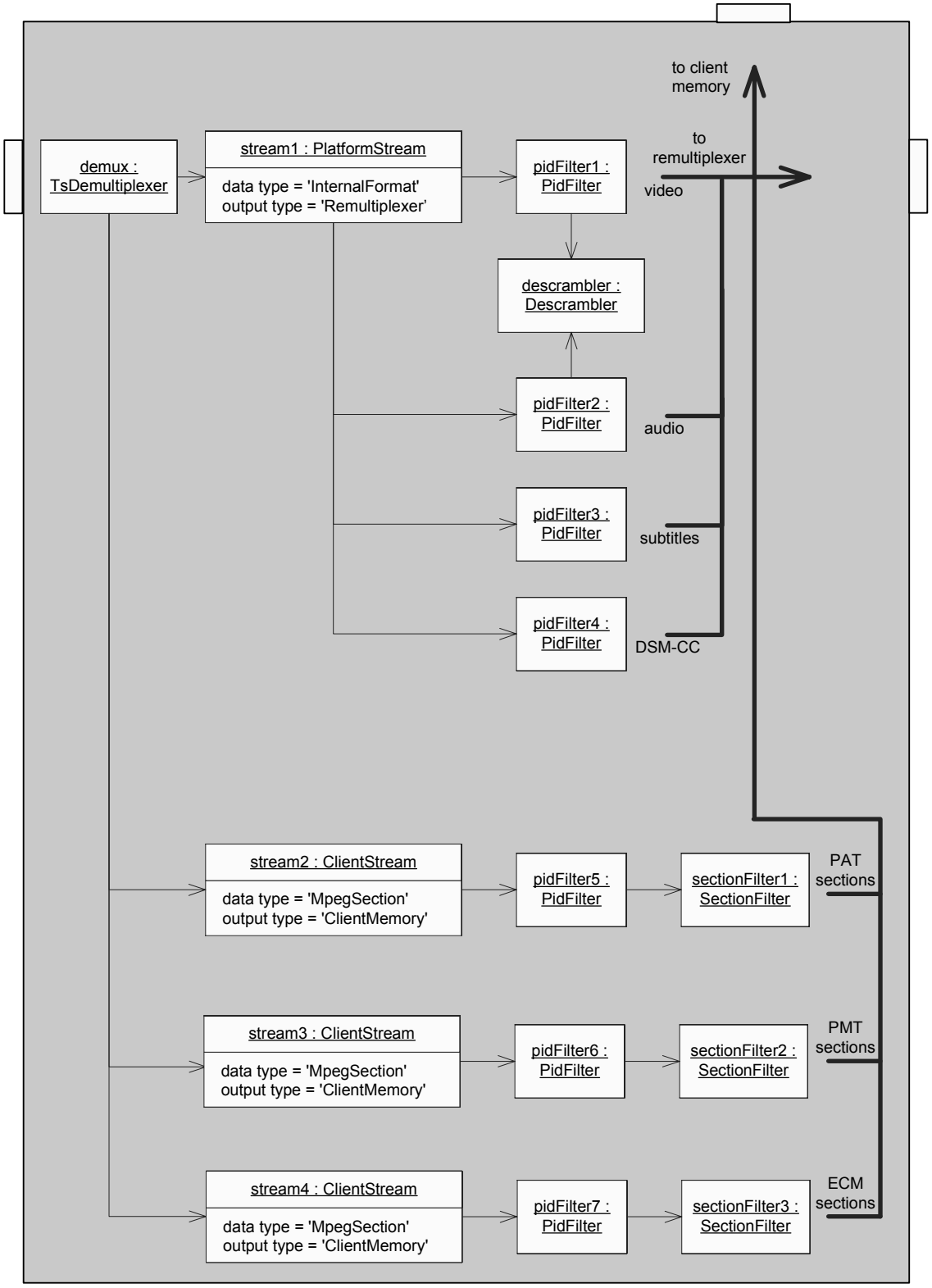


Figure 83 — SPTS generation scenario

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.1.5 Timestamped TS or SPTS

In a Transport Stream, either all of the Transport Packets (TPs) have a time stamp, or none of them. A time stamp indicates the relative timing of when a TP should be processed by the TS demultiplexer. Time stamps are based on a 27 MHz clock. See [16] for more details on time stamps.

A control can be provided by the TS demultiplexer to Clients so that they can indicate whether the input has time-stamps or not and whether, if present, they must be used. It they are not present, or if they must not be used, then the input rate of the TS determines when the TPs are demultiplexed (this is for example the case when the source of the input is a broadcast input), or the platform autonomously determines that.

Another control can be provided by the TS demultiplexer to Clients so that they can indicate whether the output SPTS has time-stamped TS packets or not. Some applications, e.g. some that conform to specific profiles of DLNA, require a time-stamped TS.

8.1.13.2 Types & Constants

8.1.13.2.1 Public Types & Constants

8.1.13.2.1.1 Error Codes

Signature

```
const uhErrorCode_t UH_ERR_TSDMX_PCR_FILTER_NOT_AVAILABLE    = 0x00000801;
const uhErrorCode_t UH_ERR_TSDMX_DESCR_NOT_AVAILABLE         = 0x00000802;
const uhErrorCode_t UH_ERR_TSDMX_PID_FILTER_NOT_AVAILABLE    = 0x00000803;
const uhErrorCode_t UH_ERR_TSDMX_TPH_FILTER_NOT_AVAILABLE    = 0x00000804;
const uhErrorCode_t UH_ERR_TSDMX_SCT_FILTER_NOT_AVAILABLE    = 0x00000805;
const uhErrorCode_t UH_ERR_TSDMX_PES_FILTER_NOT_AVAILABLE    = 0x00000806;
const uhErrorCode_t UH_ERR_TSDMX_FILTER_NOT_STARTED          = 0x00000807;
const uhErrorCode_t UH_ERR_TSDMX_TIME_STAMP_MODE_NOT_SET     = 0x00000808;
```

Qualifiers

— error-codes

Description

Definition of all non-standard errors that might be returned by the functions of the interfaces of a TS Demux.

Constants

Table 747

Name	Description
UH_ERR_TSDMX_PCR_FILTER_NOT_AVAILABLE	The allocation of a PCR Filter failed due to a lack of available PCR Filter objects.
UH_ERR_TSDMX_DESCR_NOT_AVAILABLE	The allocation of a Descrambler failed due to a lack of available Descrambler objects.
UH_ERR_TSDMX_PID_FILTER_NOT_AVAILABLE	The allocation of a PID Filter failed due to a lack of available PID Filter objects.
UH_ERR_TSDMX_TPH_FILTER_NOT_AVAILABLE	The allocation of a TPH Filter failed due to a lack of available TPH Filter objects.
UH_ERR_TSDMX_SCT_FILTER_NOT_AVAILABLE	The allocation of a Section Filter failed due to a lack of available Section Filter objects.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
UH_ERR_TSDMX_PES_FILTER_NOT_AVAILABLE	The allocation of a PES Filter/Notifier failed due to a lack of available PES Filter objects.
UH_ERR_TSDMX_FILTER_NOT_STARTED	Starting the PID or TPH Filter failed.
UH_ERR_TSDMX_TIME_STAMP_MODE_NOT_SET	The update of the time stamp mode of the TS Demux did not occur: setting the time stamp mode of the TS Demux can be effectively performed only when the TS Demux is not actively processing an input transport stream.

8.1.13.2.1.2 uhTsDmx_OutBase

Signature

```
const UInt8
uhTsDmx_OutBase = 1;
```

Qualifiers
None.

Description
Defines the constant that identifies the first output pin of a Transport Stream Demultiplexing logical component.

8.1.13.2.1.3 uhTsDmx_Notification_t

Signature

```
typedef enum _uhTsDmx_Notification_t {
    uhTsDmx_SubscriptionChanged  = 0x00000001,
    uhTsDmx_SyncLost             = 0x00000002,
    uhTsDmx_SyncLock             = 0x00000004,
    uhTsDmx_SyncInvalid          = 0x00000008,
    uhTsDmx_InputOverflow        = 0x00000010
} uhTsDmx_Notification_t, *puhTsDmx_Notification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhITsDmxNtf`. The values can be used to individually turn on and off notifications.

Values

Table 748

Name	Description
uhTsDmx_SubscriptionChanged	Represents `OnSubscriptionChanged`.
uhTsDmx_SyncLost	Represents `OnSyncLost`.
uhTsDmx_SyncLock	Represents `OnSyncLock`.
uhTsDmx_SyncInvalid	Represents `OnSyncInvalid`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTsDmx_InputOverflow	Represents OnInputOverflow.

8.1.13.2.1.4 uhTsDmx_Notifications_t

Signature
```
typedef Int32 uhTsDmx_Notifications_t, *puhTsDmx_Notifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uhTsDmx_Notification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.13.2.1.5 uhTsDmx_AllNotifications

Signature
```
const uhTsDmx_Notifications_t uhTsDmx_AllNotifications = 0x0000001F;
```

Qualifiers
None.

Description
Defines the set of all values of type uhTsDmx_Notification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.1.13.2.1.6 uhTsDmx_StreamBufferNotification_t

Signature
```
typedef enum _uhTsDmx_StreamBufferNotification_t {
    uhTsDmx_StreamBufferSubscriptionChanged  = 0x00000001,
    uhTsDmx_StreamBufferFull                 = 0x00000002
} uhTsDmx_StreamBufferNotification_t, *puhTsDmx_StreamBufferNotification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhITsDmxStreamBufferNtf. The values can be used to individually turn on and off notifications.

Values

Table 749

Name	Description
uhTsDmx_StreamBufferSubscriptionChanged	Represents OnSubscriptionChanged.
uhTsDmx_StreamBufferFull	Represents OnBufferFull.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.2.1.7 uhTsDmx_StreamBufferNotifications_t

Signature

```
typedef Int32 uhTsDmx_StreamBufferNotifications_t,
*puhTsDmx_StreamBufferNotifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type `uhTsDmx_StreamBufferNotification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.1.13.2.1.8 uhTsDmx_StreamBufferAllNotifications

Signature

```
const uhTsDmx_StreamBufferNotifications_t uhTsDmx_StreamBufferAllNotifications =
0x00000003;
```

Qualifiers

None.

Description

Defines the set of all values of type `uhTsDmx_StreamBufferNotification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.1.13.2.1.9 uhTsDmx_PcrNotification_t

Signature

```
typedef enum _uhTsDmx_PcrNotification_t {
    uhTsDmx_PcrSubscriptionChanged    = 0x00000001,
    uhTsDmx_PcrFilterStopped          = 0x00000002
} uhTsDmx_PcrNotification_t, *puhTsDmx_PcrNotification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhITsDmxPcrFilterNtf`. The values can be used to individually turn on and off notifications.

Values

Table 750

Name	Description
`uhTsDmx_PcrSubscriptionChanged`	Represents `OnSubscriptionChanged`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTsDmx_PcrFilterStopped	Represents OnStopped.

8.1.13.2.1.10 uhTsDmx_PcrNotifications_t

Signature
```
typedef Int32 uhTsDmx_PcrNotifications_t, *puhTsDmx_PcrNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uhTsDmx_PcrNotification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.13.2.1.11 uhTsDmx_PcrAllNotifications

Signature
```
const uhTsDmx_PcrNotifications_t uhTsDmx_PcrAllNotifications = 0x00000003;
```

Qualifiers
None.

Description
Defines the set of all values of type uhTsDmx_PcrNotification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.1.13.2.1.12 uhTsDmx_PidNotification_t

Signature
```
typedef enum _uhTsDmx_PidNotification_t {
    uhTsDmx_PidSubscriptionChanged    = 0x00000001,
    uhTsDmx_PidFilterStopped          = 0x00000002
} uhTsDmx_PidNotification_t, *puhTsDmx_PidNotification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhITsDmxPidFilterNtf. The values can be used to individually turn on and off notifications.

Values

Table 751

Name	Description
uhTsDmx_PidSubscriptionChanged	Represents OnSubscriptionChanged.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTsDmx_PidFilterStopped	Represents OnStopped.

8.1.13.2.1.13 uhTsDmx_PidNotifications_t

Signature
```
typedef Int32 uhTsDmx_PidNotifications_t, *puhTsDmx_PidNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uhTsDmx_PidNotification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.13.2.1.14 uhTsDmx_PidAllNotifications

Signature
```
const uhTsDmx_PidNotifications_t uhTsDmx_PidAllNotifications = 0x00000003;
```

Qualifiers
None.

Description
Defines the set of all values of type uhTsDmx_PidNotification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.1.13.2.1.15 uhTsDmx_SectionNotification_t

Signature
```
typedef enum _uhTsDmx_SectionNotification_t {
    uhTsDmx_SectionSubscriptionChanged  = 0x00000001,
    uhTsDmx_SectionFilterStopped        = 0x00000002,
    uhTsDmx_SectionArrived              = 0x00000004
} uhTsDmx_SectionNotification_t, *puhTsDmx_SectionNotification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhITsDmxSectionFilterNtf. The values can be used to individually turn on and off notifications.

Values

Table 752

Name	Description
uhTsDmx_SectionSubscriptionChanged	Represents OnSubscriptionChanged.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTsDmx_SectionFilterStopped	Represents OnStopped.
uhTsDmx_SectionArrived	Represents OnSectionArrived.

8.1.13.2.1.16 uhTsDmx_SectionNotifications_t

Signature
```
typedef Int32 uhTsDmx_SectionNotifications_t, *puhTsDmx_SectionNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uhTsDmx_SectionNotification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.13.2.1.17 uhTsDmx_SectionAllNotifications

Signature
```
const uhTsDmx_SectionNotifications_t uhTsDmx_SectionAllNotifications = 0x00000007;
```

Qualifiers
None.

Description
Defines the set of all values of type uhTsDmx_SectionNotification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.1.13.2.1.18 uhTsDmx_PesNotification_t

Signature
```
typedef enum _uhTsDmx_PesNotification_t {
    uhTsDmx_PesSubscriptionChanged    = 0x00000001,
    uhTsDmx_PesNotifierStopped        = 0x00000002,
    uhTsDmx_PesCompletePacketArrived  = 0x00000004,
    uhTsDmx_PesPartialPacketArrived   = 0x00000008,
    uhTsDmx_PesPacketCorrupt          = 0x00000010,
    uhTsDmx_PesStreamIdChanged        = 0x00000020
} uhTsDmx_PesNotification_t, *puhTsDmx_PesNotification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhITsDmxPesNotifierNtf. The values can be used to individually turn on and off notifications.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 753

Name	Description
uhTsDmx_PesSubscriptionChanged	Represents OnSubscriptionChanged
uhTsDmx_PesNotifierStopped	Represents OnStopped
uhTsDmx_PesCompletePacketArrived	Represents OnCompletePacketArrived
uhTsDmx_PesPartialPacketArrived	Represents OnPartialPacketArrived.
uhTsDmx_PesPacketCorrupt	Represents OnPacketCorrupt
uhTsDmx_PesStreamIdChanged	Represents OnStreamIdChanged

8.1.13.2.1.19 uhTsDmx_PesNotifications_t

Signature

```
typedef Int32 uhTsDmx_PesNotifications_t, *puhTsDmx_PesNotifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uhTsDmx_PesNotification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.13.2.1.20 uhTsDmx_PesAllNotifications

Signature

```
const uhTsDmx_PesNotifications_t uhTsDmx_PesAllNotifications = 0x0000003F;
```

Qualifiers

None.

Description

Defines the set of all values of type uhTsDmx_PesNotification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.2.1.21 uhTsDmx_DescramblerType_t

Signature

```
typedef enum _uhTsDmx_DescramblerType_t {
    uhTsDmx_TdesFbEcbRbNoneSbNone    = 0x00000001,
    uhTsDmx_TdesFbEcbRbOfbIvSbNone   = 0x00000002,
    uhTsDmx_TdesFbEcbRbOfbPbSbNone   = 0x00000004,
    uhTsDmx_TdesFbEcbRbOfbIvSbOfbIv  = 0x00000008,
    uhTsDmx_TdesFbEcbRbOfbPbSbOfbIv  = 0x00000010,
    uhTsDmx_TdesFbCbcRbNoneSbNone    = 0x00000020,
    uhTsDmx_TdesFbCbcRbOfbIvSbNone   = 0x00000040,
    uhTsDmx_TdesFbCbcRbOfbPbSbNone   = 0x00000080,
    uhTsDmx_TdesFbCbcRbOfbIvSbOfbIv  = 0x00000100,
    uhTsDmx_TdesFbCbcRbOfbPbSbOfbIv  = 0x00000200,
    uhTsDmx_SdesFbEcbRbNoneSbNone    = 0x00000400,
    uhTsDmx_SdesFbEcbRbOfbIvSbNone   = 0x00000800,
    uhTsDmx_SdesFbEcbRbOfbPbSbNone   = 0x00001000,
    uhTsDmx_SdesFbEcbRbOfbIvSbOfbIv  = 0x00002000,
    uhTsDmx_SdesFbEcbRbOfbPbSbOfbIv  = 0x00004000,
    uhTsDmx_SdesFbCbcRbNoneSbNone    = 0x00008000,
    uhTsDmx_SdesFbCbcRbOfbIvSbNone   = 0x00010000,
    uhTsDmx_SdesFbCbcRbOfbPbSbNone   = 0x00020000,
    uhTsDmx_SdesFbCbcRbOfbIvSbOfbIv  = 0x00040000,
    uhTsDmx_SdesFbCbcRbOfbPbSbOfbIv  = 0x00080000,
    uhTsDmx_Dvb                      = 0x00100000,
    uhTsDmx_Multi2                   = 0x00200000
} uhTsDmx_DescramblerType_t, *puhTsDmx_DescramblerType_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the different Descrambler types. The DES decryption algorithm described in [24] and [25] supports different decryption modes:

— Electronic Code Book (ECB),

— Cipher Block Chain (CBC), and

— Output FeedBack (OFB).

Both ECB and CBC modes operate on 64 bit blocks, while OFB can handle arbitrary block sizes. When DES scrambling has been applied on MPEG-2 transport packets, only the transport packet payload is encrypted.

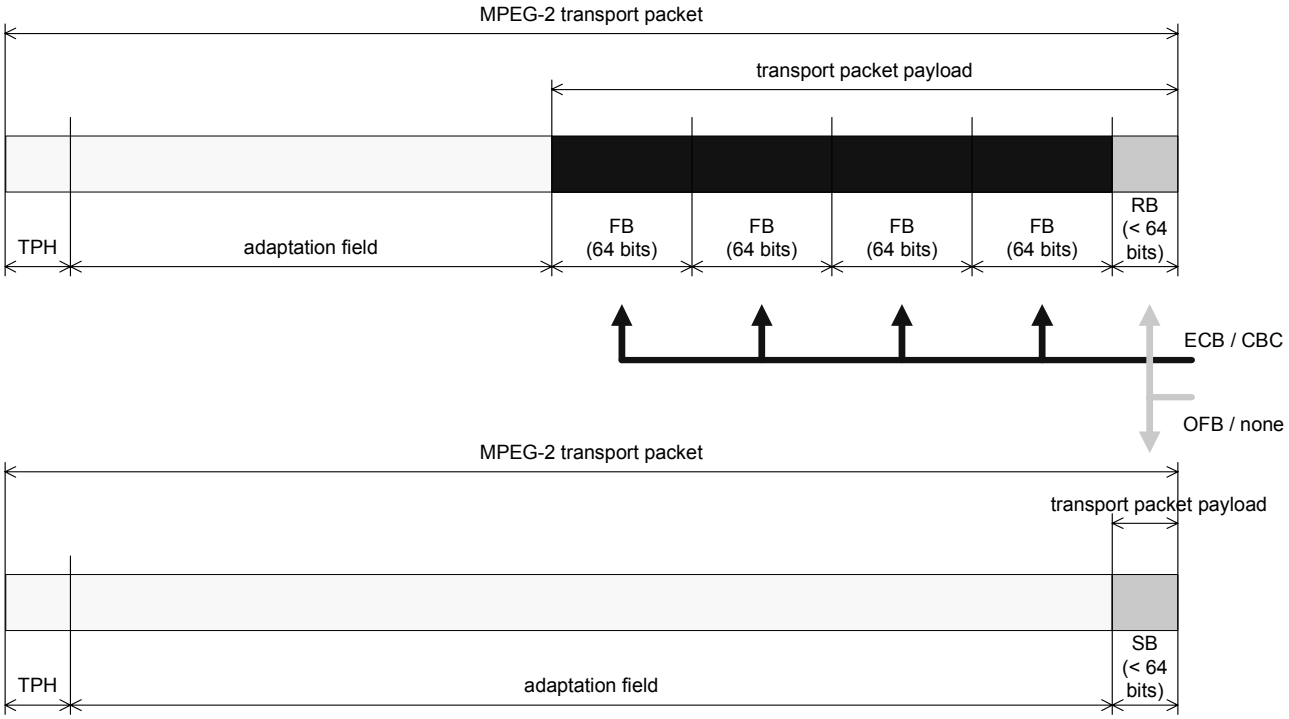
During DES scrambling, the payload (which is variable in size, as the adaptation field is never scrambled) is split up in a number of 64-bit blocks and a remainder block less than 64 bit. The 64-bit blocks (or full blocks) are typically scrambled according the ECB or CBC mode. The remainder (or residual block) is scrambled according the OFB mode or not scrambled at all.

In case the payload size is less than 64 bit, the payload (or solitary block) is scrambled according the OFB mode as well or not scrambled at all.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



FB = full block
RB = residual block
SB = solitary block

Figure 84 — DES and MPEG-2

Values

Table 754

Name	Description
`uhTsDmx_TdesFbEcbRbNoneSbNone`	Triple DES, Electronic Codebook on full blocks, none on residual blocks, none on solitary blocks.
`uhTsDmx_TdesFbEcbRbOfbIvSbNone`	Triple DES, Electronic Codebook on full blocks, Output Feedback with IV on residual blocks, none on solitary blocks.
`uhTsDmx_TdesFbEcbRbOfbPbSbNone`	Triple DES, Electronic Codebook on full blocks, Output Feedback with previous block (as IV) on residual blocks, none on solitary blocks.
`uhTsDmx_TdesFbEcbRbOfbIvSbOfbIv`	Triple DES, Electronic Codebook on full blocks, Output Feedback with IV on residual blocks, Output Feedback with IV on solitary blocks.
`uhTsDmx_TdesFbEcbRbOfbPbSbOfbIv`	Triple DES, Electronic Codebook on full blocks, Output Feedback with previous block (as IV) on residual blocks, Output Feedback with IV on solitary blocks.
`uhTsDmx_TdesFbCbcRbNoneSbNone`	Triple DES, Cipher Block Chaining on full blocks, none on residual blocks, none on solitary blocks.
`uhTsDmx_TdesFbCbcRbOfbIvSbNone`	Triple DES, Cipher Block Chaining on full blocks, Output Feedback with IV on residual blocks, none on solitary blocks.
`uhTsDmx_TdesFbCbcRbOfbPbSbNone`	Triple DES, Cipher Block Chaining on full blocks, Output Feedback with previous block (as IV) on residual blocks, none on solitary blocks.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTsDmx_TdesFbCbcRbOfbIvSbOfbIv	Triple DES, Cipher Block Chaining on full blocks, Output Feedback with IV on residual blocks, Output Feedback with IV on solitary blocks.
uhTsDmx_TdesFbCbcRbOfbPbSbOfbIv	Triple DES, Cipher Block Chaining on full blocks, Output Feedback with previous block (as IV) on residual blocks, Output Feedback with IV on solitary blocks.
uhTsDmx_SdesFbEcbRbNoneSbNone	Single DES, Electronic Codebook on full blocks, none on residual blocks, none on solitary blocks.
uhTsDmx_SdesFbEcbRbOfbIvSbNone	Single DES, Electronic Codebook on full blocks, Output Feedback with IV on residual blocks, none on solitary blocks.
uhTsDmx_SdesFbEcbRbOfbPbSbNone	Single DES, Electronic Codebook on full blocks, Output Feedback with previous block (as IV) on residual blocks, none on solitary blocks.
uhTsDmx_SdesFbEcbRbOfbIvSbOfbIv	Single DES, Electronic Codebook on full blocks, Output Feedback with IV on residual blocks, Output Feedback with IV on solitary blocks.
uhTsDmx_SdesFbEcbRbOfbPbSbOfbIv	Single DES, Electronic Codebook on full blocks, Output Feedback with previous block (as IV) on residual blocks, Output Feedback with IV on solitary blocks.
uhTsDmx_SdesFbCbcRbNoneSbNone	Single DES, Cipher Block Chaining on full blocks, none on residual blocks, none on solitary blocks.
uhTsDmx_SdesFbCbcRbOfbIvSbNone	Single DES, Cipher Block Chaining on full blocks, Output Feedback with IV on residual blocks, none on solitary blocks.
uhTsDmx_SdesFbCbcRbOfbPbSbNone	Single DES, Cipher Block Chaining on full blocks, Output Feedback with previous block (as IV) on residual blocks, none on solitary blocks.
uhTsDmx_SdesFbCbcRbOfbIvSbOfbIv	Single DES, Cipher Block Chaining on full blocks, Output Feedback with IV on residual blocks, Output Feedback with IV on solitary blocks.
uhTsDmx_SdesFbCbcRbOfbPbSbOfbIv	Single DES, Cipher Block Chaining on full blocks, Output Feedback with previous block (as IV) on residual blocks, Output Feedback with IV on solitary blocks.
uhTsDmx_Dvb	DVB descrambling.
uhTsDmx_Multi2	Multi2 descrambling.

8.1.13.2.1.22 uhTsDmx_DescramblerTypes_t

Signature

```
typedef Int32 uhTsDmx_DescramblerTypes_t,
              *puhTsDmx_DescramblerTypes_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of descrambler types.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.2.1.23 uhTsDmx_StreamDataType_t

Signature

```
typedef enum _uhTsDmx_StreamDataType_t {
    uhTsDmx_InternalFormat  = 0x00000001,
    uhTsDmx_Pes             = 0x00000002,
    uhTsDmx_Tp              = 0x00000003,
    uhTsDmx_MpegSection     = 0x00000004
} uhTsDmx_StreamDataType_t, *puhTsDmx_StreamDataType_t;
```

Qualifiers
None.

Description
Values of this type represent the Stream data types.

Values

Table 755

Name	Description
uhTsDmx_InternalFormat	Format used when data does not have to be routed to the client, but is streaming in the platform.
uhTsDmx_Pes	Packetized Elementary Stream.
uhTsDmx_Tp	Transport Packet / (Single Program) Transport Stream.
uhTsDmx_MpegSection	MPEG-2 sections.

8.1.13.2.1.24 uhTsDmx_BufferType_t

Signature

```
typedef enum _uhTsDmx_BufferType_t {
    uhTsDmx_Linear    = 0x00000001,
    uhTsDmx_Circular  = 0x00000002
} uhTsDmx_BufferType_t, *puhTsDmx_BufferType_t;
```

Qualifiers
— enum-element

Description
Values of this type represent the buffer types.

Values

Table 756

Name	Description
uhTsDmx_Linear	Linear buffer.
uhTsDmx_Circular	Circular buffer.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.2.1.25 uhTsDmx_BufferTypes_t

Signature
```
typedef Int32 uhTsDmx_BufferTypes_t, *puhTsDmx_BufferTypes_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of buffer types.

8.1.13.2.1.26 uhTsDmx_CrcMode_t

Signature
```
typedef enum _uhTsDmx_CrcMode_t {
    uhTsDmx_SkipBadSections    = 0x00000001,
    uhTsDmx_AcceptAllSections  = 0x00000002,
    uhTsDmx_DontCheckCrc       = 0x00000003
} uhTsDmx_CrcMode_t, *puhTsDmx_CrcMode_t;
```

Qualifiers
None.

Description
Values of this type represent the CRC modes.

Values

Table 757

Name	Description
uhTsDmx_SkipBadSections	A CRC check is performed autonomously by the demultiplexer. Sections with an invalid CRC value are automatically discarded. That means they will never appear in a buffer. The notification of the arrival of a section indicates the result of the CRC check.
uhTsDmx_AcceptAllSections	A CRC check is performed autonomously by the demultiplexer. All sections - regardless of the result of the CRC check - are written to a buffer. The notification of the arrival of a section indicates the result of the CRC check.
uhTsDmx_DontCheckCrc	A CRC check is not performed. All sections are written to a buffer. This is the default CRC mode.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.2.1.27 uhTsDmx_CrcStatus_t

Signature

```
typedef enum _uhTsDmx_CrcStatus_t {
    uhTsDmx_CrcOk          = 0x00000001,
    uhTsDmx_CrcNotOk       = 0x00000002,
    uhTsDmx_CrcNotChecked  = 0x00000003,
    uhTsDmx_NoCrc          = 0x00000004
} uhTsDmx_CrcStatus_t, *puhTsDmx_CrcStatus_t;
```

Qualifiers
None.

Description
Values of this type represent the CRC status.

Values

Table 758

Name	Description
uhTsDmx_CrcOk	The CRC check on the accompanying section was successful.
uhTsDmx_CrcNotOk	The CRC check on the accompanying section failed.
uhTsDmx_CrcNotChecked	The CRC check on the accompanying section has not been done yet, but it can be checked by calling the CheckCrc function of the uhITsDmxSectionFilter interface.
uhTsDmx_NoCrc	The CRC check could not be done because either the accompanying section has no CRC field or the CRC field cannot be detected because the section_syntax_indicator equals 0.

8.1.13.2.1.28 uhTsDmx_FilterMode_t

Signature

```
typedef enum _uhTsDmx_FilterMode_t {
    uhTsDmx_OneShot     = 0x00000001,
    uhTsDmx_Continuous  = 0x00000002
} uhTsDmx_FilterMode_t, *puhTsDmx_FilterMode_t;
```

Qualifiers
None.

Description
Values of this type represent the filter modes.

Values

Table 759

Name	Description
uhTsDmx_OneShot	Setting the filter mode to 'one shot' results in automatically stopping the filter as soon as one of the requested data units (e.g. a PES packet or a section) is actually filtered and signaled to the client.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTsDmx_Continuous	A filter keeps on filtering until a client explicitly stops it. This is the default filter mode.

8.1.13.2.1.29 uhTsDmx_ScramblingStatus_t

Signature

```
typedef enum _uhTsDmx_ScramblingStatus_t {
    uhTsDmx_ScramblingInvalid  = -2,
    uhTsDmx_ScramblingUnknown  = -1,
    uhTsDmx_NotScrambled       =  2,
    uhTsDmx_Scrambled          =  3
} uhTsDmx_ScramblingStatus_t, *puhTsDmx_ScramblingStatus_t;
```

Qualifiers
None.

Description
Values of this type represent the scrambling status.

Values

Table 760

Name	Description
uhTsDmx_ScramblingInvalid	The data - that is currently filtered by the PID Filter or the TPH Filter – is not temporarily available.
uhTsDmx_ScramblingUnknown	There is no data filtered by the filter. Therefore, the scrambling status is unknown.
uhTsDmx_NotScrambled	The data - that is currently filtered by the PID Filter or the TPH Filter - is not scrambled.
uhTsDmx_Scrambled	The data - that is currently filtered by the PID Filter or the TPH Filter - is scrambled.

8.1.13.2.1.30 uhTsDmx_CwType_t

Signature

```
typedef enum _uhTsDmx_CwType_t {
    uhTsDmx_OddKey         = 0x00000001,
    uhTsDmx_EvenKey        = 0x00000002,
    uhTsDmx_SystemKey      = 0x00000003,
    uhTsDmx_InitVectorCbc  = 0x00000004,
    uhTsDmx_InitVectorOfb  = 0x00000005
} uhTsDmx_CwType_t, *puhTsDmx_CwType_t;
```

Qualifiers
None.

Description
Values of this type represent the control word types.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 761

Name	Description
uhTsDmx_OddKey	Odd key.
uhTsDmx_EvenKey	Even key.
uhTsDmx_SystemKey	System key.
uhTsDmx_InitVectorCbc	Initialization vector for CBC.
uhTsDmx_InitVectorOfb	Initialization vector for OFB.

8.1.13.2.1.31 uhTsDmx_ValStat_t

Signature

```
typedef enum _uhTsDmx_ValStat_t {
    uhTsDmx_ValStatInvalid = -2,
    uhTsDmx_ValStatKnown   =  1
} uhTsDmx_ValStat_t, *puhTsDmx_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 762

Name	Description
uhTsDmx_ValStatInvalid	The property/properties are temporarily not available.
uhTsDmx_ValStatKnown	The property/properties are available and determined.

8.1.13.2.1.32 uhTsDmx_TimeStampMode_t

Signature

```
typedef enum _uhTsDmx_TimeStampMode_t {
    uhTsDmx_NonTimeStampMode    = 0,
    uhTsDmx_TimeStampMode       = 1,
    uhTsDmx_TimeStampIgnoreMode = 2
} uhTsDmx_TimeStampMode_t, *puhTsDmx_TimeStampMode_t;
```

Qualifiers
None.

Description
Values of this type represent time stamp modes.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 763

Name	Description
uhTsDmx_NonTimeStampMode	In this mode, Transport Stream packets are according to ISO 138181 TS format (Transport Stream packet size 188 bytes). Transport Stream packets do not have time stamps.
uhTsDmx_TimeStampMode	In this mode, each Transport Stream packet is preceded by 4 bytes of time stamp value (Transport Stream packet size 192 (4+188) bytes). The time stamps determine when TS packets will be de-multiplexed and sent out on the output(s) of the TsDemultiplexer. For this the time stamps are matched to an internal clock running at 27 MHz.
uhTsDmx_TimeStampIgnoreMode	In this mode, each Transport Stream packet is preceded by 4 bytes of time stamp value (Transport Stream packet size 192 (4+188) bytes), but the time stamp shall be ignored.

8.1.13.2.1.33 uhTsDmx_SyncStatus_t

Signature

```
typedef enum _uhTsDmx_SyncStatus_t {
    uhTsDmx_SyncStatusInvalid = 0,
    uhTsDmx_SyncStatusLost    = 1,
    uhTsDmx_SyncStatusLock    = 2
} uhTsDmx_SyncStatus_t, *puhTsDmx_SyncStatus_t;
```

Qualifiers

None.

Description

Values of this type represent the synchronization status of the Transport Stream Demultiplexing logical component towards its input stream.

Values

Table 764

Name	Description
uhTsDmx_SyncStatusInvalid	The synchronization status is invalid.
uhTsDmx_SyncStatusLost	The Transport Stream Demultiplexing logical component is currently not synchronized with the input stream: it did not find the synchronization byte on several consecutive transport packets of the incoming transport stream after previously being synchronized, or in the invalid lock status.
uhTsDmx_SyncStatusLock	The Transport Stream Demultiplexing logical component is currently synchronized with the input stream: it has previously encountered the synchronization byte on several consecutive transport packets of the incoming transport stream and is, currently, actively processing the incoming transport stream.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.2.2 Model Types & Constants

8.1.13.2.2.1 StreamType

Signature

```
typedef enum {
    ClientStream     = 0x00000001,
    PlatformStream   = 0x00000002
} StreamType;
```

Qualifiers
None.

Description
Values of this type represent the type of the Stream object.

Values

Table 765

Name	Description
`ClientStream`	The Stream object is a Client Stream. It routes data to the client.
`PlatformStream`	The Stream object is a Platform Stream. It streams its output data in the platform. The client is not involved in collecting data and in forwarding it to decoders and/or remultiplexers.

8.1.13.2.2.2 StreamOutputType_t

Signature

```
typedef enum {
    ClientMemory     = 1,
    VideoDecoder     = 2,
    AudioDecoder     = 3,
    TeletextDecoder  = 4,
    SubtitleDecoder  = 5,
    Remultiplexer    = 6
}
StreamOutputType_t;
```

Qualifiers
None.

Description
Values of this type represent the Stream output types. All output types - except `ClientMemory` - refer to data streaming in the platform. The client is not involved in collecting data and forwarding this data to decoders and remultiplexers.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 766

Name	Description
`ClientMemory`	Data is routed to client accessible memory.
`VideoDecoder`	Data is routed to a video decoder. The client has no access to this data.
`AudioDecoder`	Data is routed to an audio decoder. The client has no access to this data.
`TeletextDecoder`	Data is routed to a teletext decoder. The client has no access to this data.
`SubtitleDecoder`	Data is routed to a subtitle decoder. The client has no access to this data.
`Remultiplexer`	Data is routed to a remultiplexer. The client has no access to this data.

8.1.13.3 Logical Component

8.1.13.3.1 Interface-Role Model

Figure 85 and Figure 86 depict the interface-role model of the Transport Stream Demultiplexing logical component. They show the control and notification interfaces and the roles involved with the Transport Stream Demultiplexing logical component. The role `TsDemultiplexingClient`(s) represents one or more clients of the Transport Stream Demultiplexing logical component. It is drawn as one box because otherwise it might suggest a certain partitioning of client software.

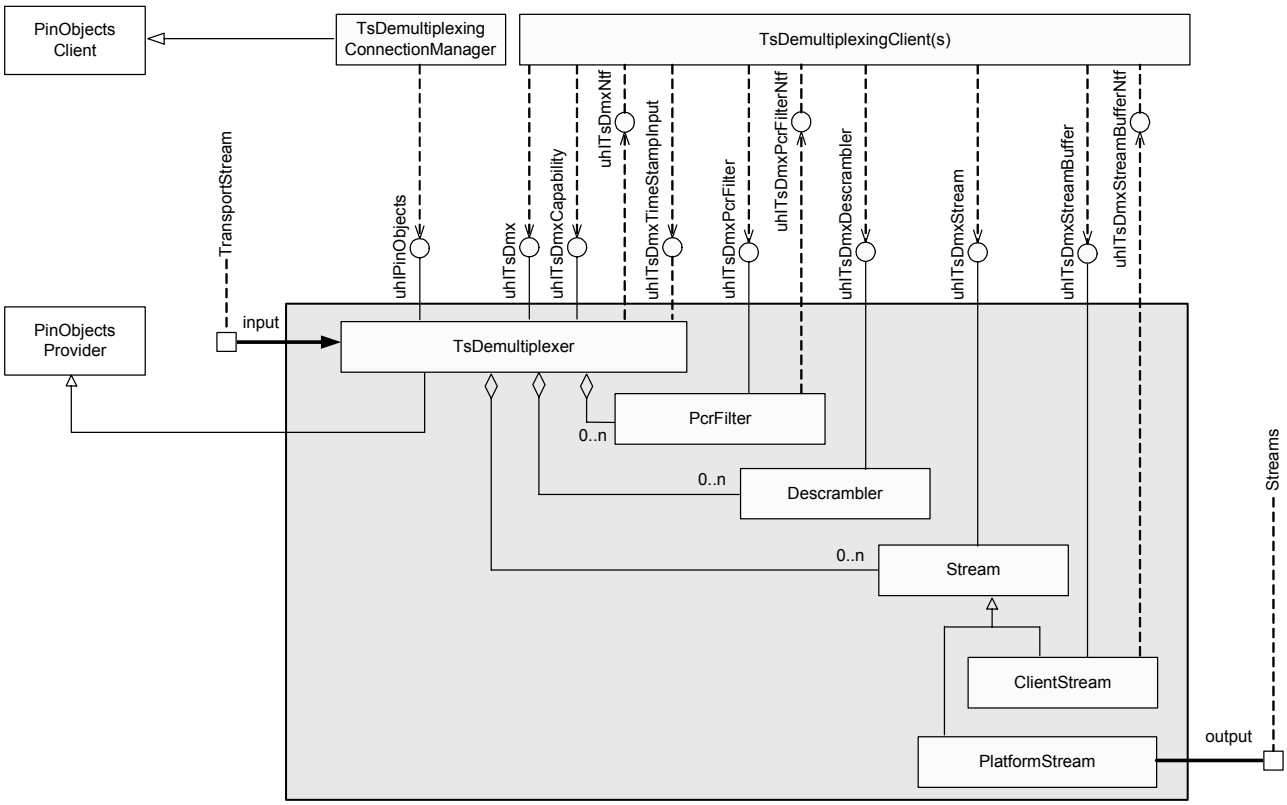


Figure 85 — Interface-Role Model (upper part)

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

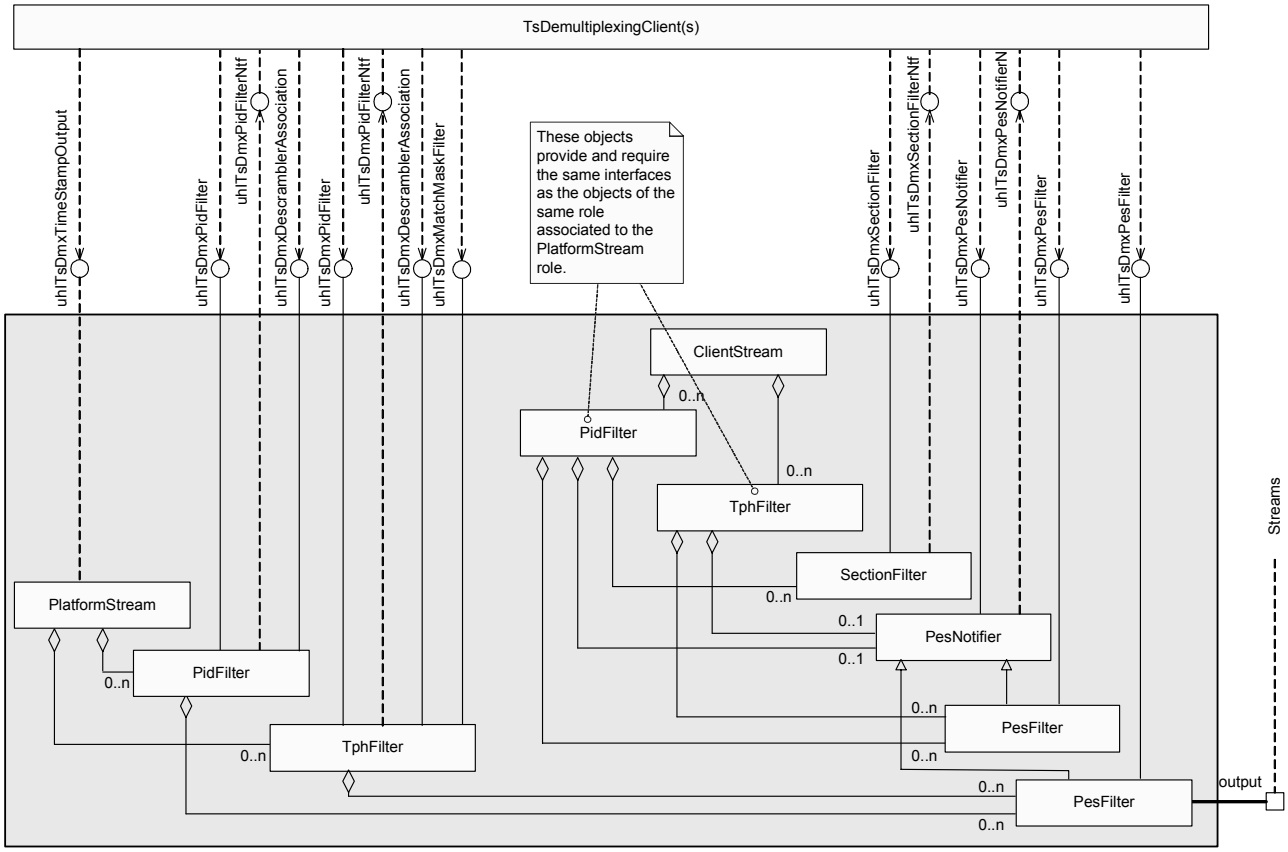


Figure 86 — Interface-Role Model (lower part)

In general, a TS Demux supports a number of filters (PCR, PID, TPH, PES, section) and descramblers. In practice, this number is dictated by the hardware. The supported number of filters and descramblers can be obtained via `uhITsDmxCapability`.

Using the `uhITsDmx` interface, PCR Filters, Descramblers and Client Streams can be allocated and the `uhITsDmxPcrFilter`, `uhITsDmxDescrambler`, or `uhITsDmxStream` interface of either a PCR Filter, Descrambler or a Client Stream can be obtained by a `TsDemultiplexingClient`.

A `TsDemultiplexingClient` can have two types of Streams. These types are represented by the `ClientStream` and the `PlatformStream` roles. These roles have common behavior which is captured in a general `Stream` role.

A `TsDemultiplexer` has an interface (`uhIPinObjects`) that maps the pin ID of an output stream to the `PlatformStream` delivering the output stream. This interface is meant for exclusive use by the connection manager (represented by the `TsDemultiplexingConnectionManager` role). The `uhITsDmxStream` interface of a Platform Stream can be obtained via the `GetPinRelatedObject` function of the `uhIPinObjects` interface; see 6.1.5.

The “base roles” `PinObjectsClient` and `PinObjectsProvider` are defined in 6.1.5.

The `uhITsDmxStreamBuffer` interface contains the functionality specific to a Client Stream (i.e. related to the buffer control). Functionality common to all Streams is provided on the `uhITsDmxStream` interface.

A `Stream` object can have two types of filters associated with it. These types are represented by the `PidFilter` and the `TphFilter` roles. These roles have common behavior. The `TphFilter` is a specialization of the `PidFilter` role.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Using the uhITsDmxStream interface, PID Filters and TPH Filters can be allocated and the uhITsDmxPidFilter interface of either a PID Filter or a TPH Filter can be obtained by a TsDemultiplexingClient.

Functionality common to PID Filters and TPH Filters is provided on the uhITsDmxPidFilter and uhITsDmxDescramblerAssociation interfaces. The latter provides functionality to associate a Descrambler to a PID Filter or to a TPH Filter. The uhITsDmxMatchMaskFilter interface contains the functionality specific to a TPH Filter.

A PidFilter object can have different types of filter associated with it. These types are represented by the SectionFilter, the PesFilter, and the PesNotifier roles. A PidFilter object can have zero or more SectionFilter, or zero or more PesFilter, or at most one PesNotifier. A TphFilter object can only have zero or more PesFilter or at most one PesNotifier associated with it.

Using the uhITsDmxPidFilter interface, Section Filters, PES Filters and PES Notifiers can be allocated and the uhITsDmxSectionFilter, uhITsDmxPesFilter, or uhITsDmxPesNotifier interface of either a Section Filter, a PES Filter, or a PES Notifier can be obtained by a TsDemultiplexingClient.

8.1.13.3.2 Diversity

8.1.13.3.2.1 Provided Interfaces

Table 767

Role	Interface	Presence
TsDemultiplexer	uhIPinObjects	mandatory
	uhITsDmx	mandatory
	uhITsDmxCapability	mandatory
	uhITsDmxTimeStampInput	optional
TsDemultiplexingClient	uhITsDmxNtf	optional
PcrFilter	uhITsDmxPcrFilter	mandatory
TsDemultiplexingClient	uhITsDmxPcrFilterNtf	optional
Descrambler	uhITsDmxDescrambler	mandatory
Stream	uhITsDmxStream	mandatory
PlatformStream	uhITsDmxTimeStampOutput	optional[20]
ClientStream	uhITsDmxStreamBuffer	mandatory
TsDemultiplexingClient	uhITsDmxStreamBufferNtf	optional
PidFilter	uhITsDmxPidFilter	mandatory
	uhITsDmxDescramblerAssociation	mandatory
TphFilter	uhITsDmxPidFilter	mandatory
	uhITsDmxDescramblerAssociation	mandatory
	uhITsDmxMatchMaskFilter	mandatory
TsDemultiplexingClient	uhITsDmxPidFilterNtf	optional
SectionFilter	uhITsDmxSectionFilter	mandatory
TsDemultiplexingClient	uhITsDmxSectionFilterNtf	optional

20 See the constraint on the presence of this interface in subclause 8.1.13.3.2.3.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Role	Interface	Presence
PesFilter	uhITsDmxPesNotifier	mandatory
	uhITsDmxPesFilter	mandatory
TsDemultiplexingClient	uhITsDmxPesNotifierNtf	optional
PesNotifier	uhITsDmxPesNotifier	mandatory

8.1.13.3.2.2 Configurable Items

Table 768

Role	Attribute
TsDemultiplexer	suppDescramblerTypes
	descramblerTypeRestriction
	suppBufferTypes
	maxNrPcrFilters
	maxNrDescramblers
	maxNrPidFilters
	maxNrTphFilters
	MaxNrSectionFilters
	maxNrPesFilters
	nrPlatformStreams
	cwDepth
	tphFilterDepth
	sectionFilterDepth
	pesFilterDepth
	pidRestriction
PlatformStream	outputType

8.1.13.3.2.3 Constraints

— (maxNrPidFilters ≥ 1) || (maxNrTphFilters ≥ 1)
— The uhITsDmxTimeStampOutput interface may only be present and provided by a PlatformStream role if the outputType attribute for this PlatformStream equals Remultiplexer.

8.1.13.3.3 Instantiation

8.1.13.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 769

Type	Object	Multiplicity
TsDemultiplexer	demux	1

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Type	Object	Multiplicity
PlatformStream	pPlatformStreams[]	demux.nrPlatformStreams

8.1.13.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— demux.pinIds == demux.outputPinIds
— For each i in {0 .. demux.nrPlatformStreams-1}
  — demux.pinObject[uhTsDmx_OutBase+i] == pPlatformStreams[i]
— demux.platformStreams == {pPlatformStreams[0] .. pPlatformStreams[demux.nrPlatformStreams-1]}
— demux.nrPcrFilters == 0
— demux.pcrFilters == {}
— For each t in uhTsDmx_DescramblerType_t
  — demux.nrDescramblers[t] == 0
— demux.descramblers == {}
— demux.nrPidFilters == 0
— demux.nrTphFilters == 0
— demux.nrSectionFilters == 0
— demux.nrPesFilters == 0
— demux.nrClientStreams == 0
— demux.clientStreams == {}
— demux.timeStampMode == uhTsDmx_NonTimeStampMode
— demux.syncLock == uhTsDmx_SyncStatusInvalid
— For each i in {0..demux.nrPlatformStreams-1}
  — pPlatformStreams[i].streamObjectType == PlatformStream
  — pPlatformStreams[i].dataType == uhTsDmx_InternalFormat
  — pPlatformStreams[i].nrPidFilters == 0
  — pPlatformStreams[i].pidFilters == {}
  — pPlatformStreams[i].nrTphFilters == 0
  — pPlatformStreams[i].tphFilters == {}
  — pPlatformStreams[i].nrTransportPackets == 0
  — pPlatformStreams[i].enableTimeStamps == False

8.1.13.3.4 Execution Constraints

The interfaces provided by the roles of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.4 Roles

8.1.13.4.1 TsDemultiplexer

Signature

```
role TsDemultiplexer specializes pinObjectsProvider {
    const uhTsDmx_DescramblerTypes_t  suppDescramblerTypes;
    const Bool                        descramblerTypeRestriction;
    const uhTsDmx_BufferTypes_t       suppBufferTypes;
    const UInt32                      maxNrPcrFilters;
    const UInt32                    maxNrDescramblers[uhTsDmx_DescramblerType_t];
    const UInt32                      maxNrPidFilters;
    const UInt32                      maxNrTphFilters;
    const UInt32                      maxNrSectionFilters;
    const UInt32                      maxNrPesFilters;
    UInt32                            nrPcrFilters;
    setof{PcrFilter}                  pcrFilters;
    UInt32                            nrDescramblers[uhTsDmx_DescramblerType_t];
    setof{Descrambler}                descramblers;
    UInt32                            nrPidFilters;
    UInt32                            nrTphFilters;
    UInt32                            nrSectionFilters;
    UInt32                            nrPesFilters;
    const UInt32                      nrPlatformStreams;
    setof{PlatformStream}             platformStreams;
    UInt32                            nrClientStreams;
    setof{ClientStream}               clientStreams;
    setof{PinId}                      outputPinIds;
    const UInt32                      cwDepth[uhTsDmx_DescramblerType_t];
    const UInt32                      tphFilterDepth;
    const UInt32                      sectionFilterDepth;
    const UInt32                      pesFilterDepth;
    const Bool                        pidRestriction;
    uhTsDmx_TimeStampMode_t           timeStampMode;
    uhTsDmx_SyncStatus_t              syncLock;
}
```

Qualifiers

— root

Description

A `TsDemultiplexer` object contains a number of `PcrFilter`, `Descrambler` and `Stream` objects for which it is the resource server. A limited number of `PcrFilter`s, `Descrambler`, `PidFilter`, `TphFilter`, `SectionFilter`, and `PesFilter` (and `PesNotifier`) objects are supported within a Transport Stream Demultiplexing logical component. A `TsDemultiplexer` object provides facilities to allocate `PcrFilter`, `Descrambler`, and `Stream` objects. It also provides facilities to query the maximum number of filters/descramblers available and the number of filters/descramblers currently available.

Independent Attributes

Table 770

Name	Description
`suppDescramblerTypes`	The descrambler types supported by the Transport Stream Demultiplexing logical component.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
descramblerTypeRestriction	Indication whether at any point in time all of the allocated descramblers have to be of the same type. It is True when at any point in time all of the allocated descramblers have to be of the same type. It is False, otherwise.
suppBufferTypes	The set of buffer types that can be handled by the platform.
maxNrPcrFilters	The maximum number of PCR Filters supported by the Transport Stream Demultiplexing logical component.
maxNrDescramblers	For each descrambler type, the maximum number of Descramblers supported by the Transport Stream Demultiplexing logical component.
maxNrPidFilters	The maximum number of PID Filters supported by the Transport Stream Demultiplexing logical component.
maxNrTphFilters	The maximum number of TPH Filters supported by the Transport Stream Demultiplexing logical component.
maxNrSectionFilters	The maximum number of Section Filters supported by the Transport Stream Demultiplexing logical component.
maxNrPesFilters	The maximum number of PES Filters and PES Notifiers supported by the Transport Stream Demultiplexing logical component.
nrPcrFilters	The number of PCR Filters actually used.
pcrFilters	The set of nrPcrFilters PCR Filters associated with the TsDemultiplexer object.
nrDescramblers	For each descrambler type, the number of Descramblers actually used.
descramblers	The set of Descramblers associated with the TsDemultiplexer object.
nrPidFilters	The number of PID Filters actually used.
nrTphFilters	The number of TPH Filters actually used.
nrSectionFilters	The number of Section Filters actually used.
nrPesFilters	The number of PES Filters and PES Notifiers actually used.
nrPlatformStreams	The number of `platform' Streams associated with the TsDemultiplexer, i.e. #platformStreams.
platformStreams	The set of `platform' Streams associated with the TsDemultiplexer.
nrClientStreams	The number of Client Streams actually used.
clientStreams	The set of Client Streams associated with the TsDemultiplexer.
cwDepth	For each type of descrambler, indicates the size in bytes (depth) of the control words.
tphFilterDepth	Size in bits (depth) of the match and mask values of TPH Filters.
sectionFilterDepth	Size in bytes (depth) of the mode, match, and mask values of Section Filters.
pesFilterDepth	Size in bytes (depth) of the match and mask values of PES Filters.
pidRestriction	Indication whether there is a restriction on using at any point in time the same PID value on more than one started PID/TPH filter. It is True when the restriction applies: using the same PID value on more than one started PID/TPH filter is not supported. It is False when the restriction does not apply: using the same PID value on more than one started PID/TPH filter is supported.
timeStampMode	The time stamp mode of the TsDemultiplexer.
syncLock	Indicates whether the Transport Stream Demultiplexing logical component has previously encountered the synchronization byte on several consecutive transport packets of the incoming transport stream and is, currently, actively processing the incoming transport stream.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes

Table 771

Name	Description
outputPinIds	The set of pin IDs of the PlatformStreams associated with the TsDemultiplexer, which is equal to { uhTsDmx_OutBase + i for which 0 ≤ i < nrPlatformStreams }.

Invariants

- 0 ≤ nrPcrFilters ≤ maxNrPcrFilters
- For each t in uhTsDmx_DescramblerType_t
  - 0 ≤ nrDescramblers[t] ≤ maxNrDescramblers[t]
- 0 ≤ nrPidFilters ≤ maxNrPidFilters
- 0 ≤ nrTphFilters ≤ maxNrTphFilters
- 0 ≤ nrSectionFilters ≤ maxNrSectionFilters
- 0 ≤ nrPesFilters ≤ maxNrPesFilters
- (maxNrPidFilters ≥ 1) || (maxNrTphFilters ≥ 1)

Instantiation

A TsDemultiplexer is always instantiated as part of a TS Demux; see subclause 8.1.13.3.3.2.

Streaming Behavior

The TsDemultiplexer continually observes the incoming transport stream of the Transport Stream Demultiplexing logical component. It continually tries to find the synchronization byte of the transport packets of the transport stream. After encountering the synchronization byte on several consecutive transport packets, the TsDemultiplexer is synchronised. If the TsDemultiplexer does not find the synchronization byte on several consecutive transport packets of the incoming transport stream (anymore), it becomes out of sync.

Active Behavior

The TsDemultiplexer has standard notification behavior (see uhITsDmxNtf for the events that can be notified), except for the events described below:

Table 772

Event	Action
The TsDemultiplexer encountered the synchronization byte on several consecutive transport packets of the incoming transport stream after previously being out of sync, or in the invalid lock status.	syncLock = uhTsDmx_SyncStatusLock Notify OnSyncLock(*)
The TsDemultiplexer did not find the synchronization byte on several consecutive transport packets of the incoming transport stream after previously being synchronized, or in the invalid lock status.	syncLock = uhTsDmx_SyncStatusLost Notify OnSyncLost(*)

Remarks

- The constant attributes pinIds and pinObject are inherited from the pinObjectsProvider role defined in 6.1.5 which specifies no values for these attributes.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.4.2 PcrFilter

Signature

```
role PcrFilter {
    Uint16   pid;
    Bool     started;
}
```

Qualifiers
None.

Description
The function of a `PcrFilter` is to filter Program Clock Reference (PCR) data from the input transport stream and route it to a System Time Clock recovery mechanism in order to keep the timing of the various decoders synchronized.

Independent Attributes

Table 773

Name	Description
`pid`	PID of the PCR Filter.
`started`	Indicates whether the PCR Filter is started. It is `True` if and only if the PCR Filter is started.

Dependent Attributes
None.

Invariants

`0x0000 ≤ pid ≤ 0x1FFF`

Instantiation
A `PcrFilter` is instantiated using the `AllocatePcrFilter` function of the `uhITsDmx` interface.

Streaming Behavior
As long as the `PcrFilter` is stopped, `started` is set to `False` and the `PcrFilter` does not filter Program Clock Reference (PCR) data from the output transport stream of the TsDemultiplexer.

When the client wants to start the `PcrFilter`, it can use the `Start` function of the `uhITsDmxPcrFilter` interface. As a consequence, `started` is set to True and the `PcrFilter` starts filtering PCRs on the given `pid` from the input transport stream.

Once filtered, PCR data is routed to a System Time Clock recovery mechanism.

If the client wants to stop the `PcrFilter`, it can use the `Stop` function of the `uhITsDmxPcrFilter` interface. When calling this function, the `PcrFilter` will stop as soon as possible. A notification `OnStopped` is sent to the client, if subscribed, when the `PcrFilter` has stopped, `started` is then set to False.

Active Behavior
A `PcrFilter` has standard notification behavior; see `uhITsDmxPcrFilterNtf` for the events that can be notified.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.4.3 Descrambler

Signature

```
role Descrambler {
    const uhTsDmx_DescramblerType_t   descramblerType;
    UInt8*                            cw;
    uhTsDmx_CwType_t                  cwType;
    Bool                              defaultCwUsed;
}
```

Qualifiers
None.

Description
A `Descrambler` can descramble streams whose content has been protected by the content provider through scrambling. A `Descrambler` can work for multiple streams concurrently and at various levels (TS, PES, and section). Descrambling is effective if the `Descrambler` has the right descrambling keys. A `Descrambler` provides facilities to set control words to be used for descrambling.

Independent Attributes

Table 774

Name	Description
`descramblerType`	The data type of the Descrambler.
`cw`	Points to the control word with `demux.cwDepth[descramblerType]` bytes of valid data.
`cwType`	The type of the control word.
`defaultCwUsed`	Indicates whether to use the default control word for descrambling.

Dependent Attributes
None.

Invariants

None.

Instantiation
A `Descrambler` is instantiated using the `AllocateDescrambler` function of the `uhITsDmx` interface.

Streaming Behavior
For each PID and/or TPH Filter with which the `Descrambler` is associated, the `Descrambler` descrambles data of the stream associated with the filter as long as the `Descrambler` has the right descrambling keys (control words) for this particular stream.

Active Behavior
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.4.4 Stream

Signature

```
role Stream {
    const StreamType                  streamObjectType;
    const uhTsDmx_StreamDataType_t    dataType;
    StreamOutputType_t                outputType;
    UInt32                            nrPidFilters;
    setof{PidFilter}                  pidFilters;
    UInt32                            nrTphFilters;
    setof{TphFilter}                  tphFilters;
    UInt32                            nrTransportPackets;
}
```

Qualifiers
None.

Description
A `Stream` models a stream of data of a particular type. Typically, a `Stream` has one PID or one TPH Filter. It only has more PID or TPH Filters when a Stream is used to filter a Single Program Transport Stream (SPTS), e.g. for remultiplexing. A `Stream` provides facilities to associate PID and/or TPH Filter(s).

Independent Attributes

Table 775

Name	Description
`streamObjectType`	The type of the `Stream` object.
`dataType`	The data type of the `Stream` object.
`outputType`	The output type of the `Stream` object.
`nrPidFilters`	The number of PID Filters associated with the `Stream` object.
`pidFilters`	The set of `nrPidFilters` PID Filters associated with the `Stream` object.
`nrTphFilters`	The number of TPH Filters associated with the `Stream` object.
`tphFilters`	The set of `nrTphFilters` TPH Filters associated with the `Stream` object.
`nrTransportPackets`	The number of transport packets that passed the PID Filter condition set on a PID or TPH Filter object associated to the `Stream` object since the last time the `GetNrTransportPackets` function of the `uhITsDmxStream` interface was called. For SPTS generation, the number of transport packets that passed the union of the PID Filter conditions of all PID and TPH Filters associated with the `Stream` object since the last time this function was called.

Dependent Attributes
None.

Invariants

— Only some combinations of stream data type and stream output type are supported. In Figure 87, the combinations marked with `Y' are valid, the combinations with `N' are invalid.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	uhTsDmx_Pes	uhTsDmx_Tp	uhTsDmx_MpegSection	uhTsDmx_InternalFormat
ClientMemory	Y	Y	Y	N
VideoDecoder	N	N	N	Y
AudioDecoder	N	N	N	Y
TeletextDecoder	N	N	N	Y
SubtitleDecoder	N	N	N	Y
Remultiplexer	N	N	N	Y

Figure 87 — Supported stream data type/stream output type combinations

— If `dataType == uhTsDmx_InternalFormat`
  — `streamObjectType == PlatformStream`
— else
  — `streamObjectType == ClientStream`
— Figure 88 lists the number of PID plus TPH Filters that can be allocated per `Stream` object of a certain stream output and data type.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	uhTsDmx_Pes	uhTsDmx_Tp	uhTsDmx_MpegSection	uhTsDmx_InternalFormat
ClientMemory	1	>=1	1	0
VideoDecoder	0	0	0	1
AudioDecoder	0	0	0	1
TeletextDecoder	0	0	0	1
SubtitleDecoder	0	0	0	1
Remultiplexer	0	0	0	>=1

Figure 88 — Number of PID plus TPH Filters allowed per Stream object per data type/ output type combination.

— `(nrPidFilters + nrTphFilters) ≥ 1`

Instantiation
A `Stream` is either a `ClientStream` or a `PlatformStream`. A `ClientStream` is instantiated using the `AllocateStream` function of the `uhITsDmx` interface. A `PlatformStream` is always instantiated as part of a TS Demux; see subclause 8.1.13.3.3.2.

Streaming Behavior
A `Stream` models a stream of data. The `TsDemultiplexer` provides the incoming transport stream. The input data is filtered by the filters (PID, TPH, PES, section) associated with this `Stream` and one buffer is being used per `Stream` for all data that is being filtered by the filters. A `Stream` always has one buffer to store all its output data. This buffer is not explicitly modeled.

When the `outputType` of the `Stream` is `ClientMemory`, the filtered data is routed to a client. The client has to manage the buffer explicitly.

When the `outputType` of the `Stream` is not `ClientMemory`, the filtered data is routed to the output of the TS Demux to be decoded by an appropriate decoder or remultiplexed.

Active Behavior
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.4.5 ClientStream

Signature

```
role ClientStream specializes Stream {
    uhTsDmx_BufferType_t  bufferType;
    UInt32                bufferSize;
    UInt32                bufferGranularity;
    pUInt8                pStartAddressBuffer;
    pUInt8                pEndAddressBuffer;
}
```

Qualifiers
None.

Description
A `ClientStream` is a specialization of the `Stream` role. It is a Stream for which the data is actually being routed to a client. The data of all filters attached to the Stream end up in a single buffer. The client has to manage the buffer explicitely. A `ClientStream` provides facilities to control its associated buffer.

Independent Attributes

Table 776

Name	Description
`bufferType`	The type of the buffer associated with the Client Stream object.
`bufferSize`	The size - in bytes - of the buffer associated with the Client Stream object.
`bufferGranularity`	The granularity - in bytes - of the buffer associated with the Client Stream object. The granularity specifies the rate at which packets will be notified. For example for transport packets, a granularity of 1880 bytes means a notification will be sent whenever 10 transport packets are received.
`pStartAddressBuffer`	Start address of the buffer associated with the Client Stream object.

Dependent Attributes

Table 777

Name	Description
`pEndAddressBuffer`	End address of the buffer associated with the Client Stream object

Invariants
- See Section Invariants of the `Stream` role.
- `bufferType` in `suppBufferTypes`
- `streamObjectType == ClientStream`
- `outputType == ClientMemory`
- Only buffers that are associated to a `ClientStream` object with data type `uhTsDmx_Pes` and `uhTsDmx_Tp` can have a granularity different from `0`.

Instantiation
A `ClientStream` is instantiated using the `AllocateStream` function of the `uhITsDmx` interface.

Streaming Behavior
See Section Streaming Behavior of the `Stream` role.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The filtered data is routed to a client. The client has to manage the buffer explicitly.

Active Behavior
A `ClientStream` has standard notification behavior; see `uhITsDmxStreamBufferNtf` for the events that can be notified.

Remarks
- For PES data, the granularity specifies the size of a partial PES packet that will be notified. Partial PES packets will be notified by the `uhTsDmx_PesPartialPacketArrived` event.
- When `bufferGranularity == 0`, only complete PES packets can be notified through the `OnCompletePacketArrived` notification of the `uhITsDmxPesNotifierNtf` notification interface, no (partial) PES packets will be notified through the `OnPartialPacketArrived` notification.
- The minimum granularity for transport packets equals the transport packet size.

8.1.13.4.6 PlatformStream

Signature
```
role PlatformStream specializes Stream {
    Bool       enableTimeStamps;
}
```

Qualifiers
None.

Description
A `PlatformStream` is a specialization of the `Stream` role. It is a Stream for which the data is routed to an appropriate (with respect to the type of data, e.g. audio decoder for audio data) decoder or a remultiplexer.

Independent Attributes

Table 778

Name	Description
`enableTimeStamps`	Indicates whether time stamps are enabled or disabled on the output stream of this platform stream. It is `True` if time Stamps are enabled. It is `False` otherwise.

Dependent Attributes
None.

Invariants
- See Section Invariants of the `Stream` role.
- `streamObjectType == PlatformStream`
- `outputType != ClientMemory`

Instantiation
A `PlatformStream` is always instantiated as part of a TS Demux; see subclause 8.1.13.3.3.2.

Streaming Behavior
See Section Streaming Behavior of the `Stream` role.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The filtered data is routed to the output of the TS Demux to be decoded by an appropriate decoder or remultiplexed.

If (`enableTimeStamps == True`) then each outputted TS packet is preceded by a 4-bytes time stamp. This time stamp is based on a 27 MHz clock.

Active Behavior
None.

8.1.13.4.7 PidFilter

Signature

```
role PidFilter {
    Uint16                      pid;
    Bool                        started;
    UInt32                      nrSectionFilters;
    setof{SectionFilter}        sectionFilters;
    UInt32                      nrPesFilters;
    setof{PesFilter}            pesFilters;
    puhITsDmxPesNotifier        pIPesNotifier;
    uhTsDmx_ScramblingStatus_t  scramblingStatus;
    puhITsDmxDescrambler        pIDescrambler;
}
```

Qualifiers
None.

Description
A `PidFilter` extracts transport packets with a given PID from the input transport stream. A `PidFilter` can have specialized filters attached to do additional filtering on its output stream. The following specialized filters can be associated with a `PidFilter`:

— PES Filters to filter PES packets,

— Section Filters for filtering of MPEG-2 sections.

Moreover, a `PidFilter` can have an association with one PES Notifier. However, a `PidFilter` can only be associated with one type of filter / notifier at a time. So, a `PidFilter` can be associated with either multiple Section Filters, or multiple PES Filters, or one PES Notifier.

Since the content providers can protect content through scrambling, a `PidFilter` can be associated with a `Descrambler` to descramble the appropriate data from the input transport stream.

Independent Attributes

Table 779

Name	Description
`pid`	PID of the PID Filter.
`started`	Indicates whether the PID Filter is started. It is True if and only if the PID Filter is started.
`nrSectionFilters`	The number of Section Filters associated with the PID Filter object.
`sectionFilters`	The set of `nrSectionFilters` Section Filters associated with the PID Filter object.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
nrPesFilters	The number of PES Filters associated with the PID Filter object.
pesFilters	The set of nrPesFilters PES Filters associated with the PID Filter object.
pIPesNotifier	Interface pointer of a PES Notifier associated with the PID Filter object. It is Null when no PES Notifier is associated.
scramblingStatus	Indicates whether the current stream that is filtered by the PID Filter is scrambled.
pIDescrambler	Interface pointer of a Descrambler associated with the PID Filter object. It is Null if and only if no Descrambler is associated.

Dependent Attributes
None.

Invariants

- 0x0000 ≤ pid ≤ 0x1FFF
- 0 ≤ nrSectionFilters ≤ demux. maxNrSectionFilters
- 0 ≤ nrPesFilters ≤ demux.maxNrPesFilters
- If nrSectionFilters != 0
  - nrPesFilters == 0
  - pIPesNotifier == Null
- If nrPesFilters != 0
  - nrSectionFilters == 0
  - pIPesNotifier == Null
- If pIPesNotifier != Null
  - nrSectionFilters == 0
  - nrPesFilters == 0
- If the Stream object - that is associated with the PID Filter object - does not have (data type uhTsDmx_MpegSection and output type ClientMemory), then nrSectionFilters == 0.
- If the Stream object - that is associated with the PID Filter object - does not have:
  - data type uhTsDmx_Pes and output type ClientMemory or
  - data type uhTsDmx_InternalFormat and (output type VideoDecoder or AudioDecoder or TeletextDecoder or SubtitleDecoder),

  then nrPesFilters == 0.
- If the Stream object - that is associated with the PID Filter object - does not have (data type uhTsDmx_Pes and output type ClientMemory), then pIPesNotifier == Null.

Instantiation
A PidFilter is instantiated using the AllocatePidFilter function of the uhITsDmxStream interface.

Streaming Behavior
As long as the PidFilter is stopped, started is set to False and the PidFilter does not extract transport packets with a given PID from the input transport stream of the Stream with which it is associated.

When the client wants to start the PidFilter, it can use the Start function of the uhITsDmxPidFilter interface. As a consequence, started is set to True and the PidFilter starts extracting transport packets with a given PID from the input transport stream.

Once filtered, the data is routed:

- to Section Filter(s), PES Filter(s), or a PES Notifier associated with the PidFilter;

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— otherwise to a buffer that is explicitely managed by the client if the `dataType` of the Stream is not uh`TsDmx_InternalFormat`;

— otherwise to the output of the TS Demux (to be decoded by an appropriate decoder or to be remultiplexed), if the `dataType` of the Stream is uh`TsDmx_InternalFormat`.

If the client wants to stop the `PidFilter`, it can use the `Stop` function of the `uhITsDmxPidFilter` interface. When calling this function, the `PidFilter` will stop as soon as possible, and the data associated with this PID filter (in the buffer used by the Stream object associated with this PID Filter) is flushed if the Stream is a Platform Stream (i.e. `stream.StreamType == PlatformStream`). A notification `OnStopped` is sent to the client, if subscribed. When the `PidFilter` has stopped, `started` is then set to `False`.

Active Behavior
A `PidFilter` has standard notification behavior; see `uhITsDmxPidFilterNtf` for the events that can be notified.

8.1.13.4.8 TphFilter

Signature
```
role TphFilter {
    Uint16                    pid;
    Bool                      started;
    UInt32                    nrPesFilters;
    setof{PesFilter}          pesFilters;
    puhITsDmxPesNotifier      pIPesNotifier;
    uhTsDmx_ScramblingStatus_t scramblingStatus;
    puhITsDmxDescrambler      pIDescrambler;
    UInt8                     pMatchValue[];
    UInt8                     pMaskValue[];
}
```

Qualifiers
None.

Description
A `TphFilter` extracts transport packets based on values of fields of the header and allows to set filter criteria on some other fields than the PID. A `TphFilter` can also have specialized filters attached to do additional filtering on its output stream. Only PES Filters can be associated with a `TphFilter`. Moreover, a `TphFilter` can have an association with one PES Notifier. However, a `TphFilter` can only be associated with one filter / notifier type at a time. So, a `TphFilter` can be associated with either multiple PES Filters or one PES Notifier.

Since the content providers can protect the content of programs through scrambling, a `TphFilter` can be associated with a `Descrambler` to descramble the appropriate transport packets from the input transport stream.

Independent Attributes

Table 780

Name	Description
`pid`	PID of the TPH Filter.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
started	Indicates whether the TPH Filter is started. It is True if and only if the TPH Filter is started.
nrPesFilters	The number of PES Filters associated with the TPH Filter object.
pesFilters	The set of nrPesFilters PES Filters associated with the TPH Filter object.
pIPesNotifier	Interface pointer of a PES Notifier associated with the TPH Filter object. It is Null when no PES Notifier is associated.
scramblingStatus	Indicates whether the current stream that is filtered by the TPH Filter is scrambled.
pIDescrambler	Interface pointer of a Descrambler associated with the TPH Filter object. It is Null if and only if no Descrambler is associated.
pMatchValue	Match value.
pMaskValue	Mask value.

Dependent Attributes
None.

Invariants

- 0x0000 ≤ pid ≤ 0x1FFF
- 0 ≤ nrPesFilters ≤ demux.maxNrPesFilters
- If nrPesFilters != 0
  - pIPesNotifier == Null
- If pIPesNotifier != Null
  - nrPesFilters == 0
- If the Stream object - that is associated with the TPH Filter object - does not have:
  - data type uhTsDmx_Pes and output type ClientMemory or
  - data type uhTsDmx_InternalFormat and (output type VideoDecoder or AudioDecoder or TeletextDecoder or SubtitleDecoder),

  then nrPesFilters == 0.
- If the Stream object - that is associated with the TPH Filter object - does not have (data type uhTsDmx_Pes and output type ClientMemory), then pIPesNotifier == Null.

Instantiation
A TphFilter is instantiated using the AllocateTphFilter function of the uhITsDmxStream interface.

Streaming Behavior
As long as the TphFilter is stopped, started is set to False and the TphFilter does not extract transport packets from the input transport stream of the Stream with which it is associated.

When the client wants to start the TphFilter, it can use the Start function of the uhITsDmxPidFilter interface. As a consequence, started is set to True and the TphFilter starts extracting transport packets from the input transport stream.

Once filtered, the data is routed:

- to PES Filters(s) or a PES Notifier associated with the TphFilter;
- otherwise to a buffer that is explicitely managed by the client if the dataType of the Stream is not uhTsDmx_InternalFormat;
- otherwise to the output of the TS Demux (to be decoded by an appropriate decoder or to be remultiplexed), if the dataType of the Stream is uhTsDmx_InternalFormat.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the client wants to stop the TphFilter, it can use the Stop function of the uhITsDmxPidFilter interface. When calling this function, the TphFilter will stop as soon as possible and the data associated with this TPH filter (in the buffer used by the Stream object associated with this TPH Filter) is flushed if the Stream is a Platform Stream (i.e. stream.StreamType == PlatformStream). A notification OnStopped is sent to the client, if subscribed. When the TphFilter has stopped, started is then set to False.

Active Behavior
A TphFilter has standard notification behavior; see uhITsDmxPidFilterNtf for the events that can be notified.

8.1.13.4.9 SectionFilter

Signature

```
role SectionFilter {
    Bool                        started;
    uhTsDmx_FilterMode_t        filterMode;
    UInt8                       pModeValue[demux.sectionFilterDepth];
    UInt8                       pMatchValue[demux.sectionFilterDepth];
    UInt8                       pMaskValue[demux.sectionFilterDepth];
    uhTsDmx_CrcMode_t           crcMode;
    uhTsDmx_CrcStatus_t         crc;
}
```

Qualifiers
None.

Description
A SectionFilter filters sections from the output stream of the ClientStream with which it is associated. It provides facilities to control the filter mode and allows match/mask control on the filter. A SectionFilter also provides facilities to check the integrity of data.

Independent Attributes

Table 781

Name	Description
started	Indicates whether the Section Filter is started. It is True if and only if the Section Filter is started.
filterMode	Filter mode of the Section Filter.
pModeValue	Mode value.
pMatchValue	Match value.
pMaskValue	Mask value.
crcMode	The CRC checking mode of the Section Filter.
crc	CRC status of the CRC check on the last provided section from a client.

Dependent Attributes
None.

Invariants

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Instantiation
A `SectionFilter` is instantiated using the `AllocateSectionFilter` function of the `uhITsDmxPidFilter` interface.

Streaming Behavior
As long as the `SectionFilter` is stopped, `started` is set to `False` and the `SectionFilter` does not filter sections from the output stream of the `ClientStream` with which it is associated.

When the client wants to start the `SectionFilter`, it can use the `Start` function of the `uhITsDmxSectionFilter` interface. As a consequence, `started` is set to `True` and the `SectionFilter` starts filtering sections - provided that the PID or TPH Filter it is associated with is also started - and starts signaling sections extraction in the following way. If clients have subscribe to the appropriate notifications of events that occur in a Section Filter, a notification `OnSectionArrived` is signaled when a complete MPEG section has been filtered and written to a buffer that is explicitly managed by the client.

If the client wants to stop the `SectionFilter`, it can use the `Stop` function of the `uhITsDmxSectionFilter` interface. When calling this function, the `SectionFilter` will stop as soon as possible. A notification `OnStopped` is sent to the client, if subscribed, when the `SectionFilter` has stopped, `started` is then set to `False`.

Active Behavior
A `SectionFilter` has standard notification behavior; see `uhITsDmxSectionFilterNtf` for the events that can be notified.

8.1.13.4.10 PesNotifier

Signature
```
role PesNotifier {
    Bool                 started;
    uhTsDmx_FilterMode_t filterMode;
}
```

Qualifiers
None.

Description
A `PesNotifier` signals the reception of PES packets from the output stream of the `ClientStream` with which it is associated. It provides facilities to control the filter mode. A `PesNotifier` also provides facilities to subscribe and unsubscribe to notifications on events related to the arrival of PES packets.

Independent Attributes

Table 782

Name	Description
`started`	Indicates whether the PES Notifier is started. It is True if and only if the PES Notifier is started.
`filterMode`	Filter mode of the PES Notifier.

Dependent Attributes
None.

Invariants

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

None.

Instantiation
A PesNotifier is instantiated using the AllocatePesNotifier function of the uhITsDmxPidFilter interface.

Streaming Behavior
As long as the PesNotifier is stopped, started is set to False and the PesNotifier does not signal the reception of PES packets from the output stream of the ClientStream with which it is associated.

When the client wants to start the PesNotifier, it can use the Start function of the uhITsDmxPesNotifier interface. As a consequence started is set to True and the PesNotifier starts signaling the reception of PES packets - provided that the PID or TPH Filter it is associated with is also started - in the following way if clients are subscribe to the appropriate notifications of events that occur in a PES Notifier:

— A notification OnCompletePacketArrived is signaled when a complete PES packet has been captured and written to a buffer that is explicitly managed by the client.

— A notification OnPartialPacketArrived is signaled when a part of a PES packet has been captured and written to a buffer that is explicitly managed by the client.

— A notification OnPacketCorrupt is signaled when a partial or a complete PES packet has been flagged as corrupt due to a corruption of at least one transport packet part of this packet.

— A notification OnStreamIdChanged is signaled when the `stream_id' has changed. When the PesNotifier starts, this event will be fired as well on the arrival of the first PES packet.

If the client wants to stop the PesNotifier, it can use the Stop function of the uhITsDmxPesNotifier interface. When calling this function, the PesNotifier will stop as soon as possible. A notification OnStopped is sent to the client, if subscribed, when the PesNotifier has stopped, started is then set to False.

Active Behavior
A PesNotifier has standard notification behavior; see uhITsDmxPesNotifierNtf for the events that can be notified.

8.1.13.4.11 PesFilter

Signature
```
role PesFilter specializes PesNotifier {
    UInt8          pMatchValue [demux.pesFilterDepth];
    UInt8          pMaskValue[demux.pesFilterDepth];
    uhTimeStamp_t  minDts;
    uhTimeStamp_t  maxDts;
    uhTimeStamp_t  minPts;
    uhTimeStamp_t  maxPts;
}
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
A PesFilter filters PES packets from the output stream of the Stream with which it is associated. It provides facilities for match/mask control on the filter. A PesFilter also provides facilities to control DTS/PTS ranges that are used as a filter criterion.

Independent Attributes

Table 783

Name	Description
pMatchValue	Match value.
pMaskValue	Mask value.
minDts	The minimum DTS value of the current range for DTS filtering of the PES Filter object.
maxDts	The maximum DTS value of the current range for DTS filtering of the PES Filter object.
minPts	The minimum PTS value of the current range for PTS filtering of the PES Filter object.
maxPts	The maximum PTS value of the current range for PTS filtering of the PES Filter object.

Dependent Attributes
None.

Invariants

— minDts.timeStampHigh ≤ maxDts.timeStampHigh
— If minDts.timeStampHigh == maxDts.timeStampHigh
  — minDts.timeStampLow ≤ maxDts.timeStampLow
  — minPts.timeStampHigh ≤ maxPts.timeStampHigh
— If minPts.timeStampHigh == maxPts.timeStampHigh
  — minPts.timeStampLow ≤ maxPts.timeStampLow

Instantiation
A PesFilter is instantiated using the AllocatePesFilter function of the uhITsDmxPidFilter interface.

Streaming Behavior
See Section *Streaming Behavior* of the PesNotifier role.

Active Behavior
A PesFilter has standard notification behavior; see uhITsDmxPesNotifierNtf for the events that can be notified.

8.1.13.4.12 TsDemultiplexingClient

Signature
```
role TsDemultiplexingClient {}
```

Qualifiers
— actor

Description
A TS Demultiplexing Client represents a user or users of one or more of the following interfaces: uhITsDmx, uhITsDmxCapability, uhITsDmxDescrambler, uhITsDmxStream,

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

uhITsDmxStreamBuffer, uhITsDmxPcrFilter, uhITsDmxPidFilter, uhITsDmxDescramblerAssociation, uhITsDmxMatchMaskFilter, uhITsDmxSectionFilter, uhITsDmxPesFilter, and uhITsDmxPesNotifier interfaces.

8.1.13.4.13 TsDemultiplexingConnectionManager

Signature
```
role TsDemultiplexingConnectionManager specializes PinObjectsClient {}
```

Qualifiers
— actor

Description
A TS Demultiplexing Connection Manager represents a user of the uhIPinObjects interface. This will normally be the connection manager responsible for establishing signal paths between logical components.

8.1.13.5 Interfaces

8.1.13.5.1 TsDemultiplexer::uhIPinObjects

Qualifiers
None.

Description
This interface of a TsDemultiplexer is a specialization of the uhIPinObjects interface as defined in 6.1.5. The specialization is defined by the specific values of the pinIds and the pinObject attributes of the TsDemultiplexer. These attributes are inherited from PinObjectsProvider and their values are set at instantiation time of a TsDemultiplexer; see subclause 8.1.13.3.3.2.

Interface ID
See 6.1.5.

8.1.13.5.2 uhITsDmx

Qualifiers
None.

Description
This interface of a TsDemultiplexer provides facilities to allocate PCR Filters, Streams, and Descramblers. It also provides a Subscribe and an Unsubscribe function for notifications of events related to synchronization within the TsDemultiplexer.

Interface ID
```
uuid( A9FD1A17-7E83-42A9-A9B9-98E1AABEF2D3 )
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.2.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhITsDmxNtf             *pINotify,
    [in] UInt32                  cookie,
    [in] uhTsDmx_Notifications_t notifs );
```

Qualifiers

— subscribe-function

8.1.13.5.2.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhITsDmxNtf              *pINotify,
    [in] UInt32                  cookie,
    [in] uhTsDmx_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.13.5.2.3 AllocatePcrFilter

Signature

```
uhErrorCode_t AllocatePcrFilter ( [out] uhITsDmxPcrFilter **ppIPcrFilter );
```

Qualifiers

— synchronous

— thread-safe

Description

Allocates a PCR Filter object and returns a pointer to the `uhITsDmxPcrFilter` interface of the allocated PCR Filter object.

Parameters

Table 784

Name	Description
`ppIPcrFilter`	Output parameter returning a pointer to the `uhITsDmxPcrFilter` interface of the allocated PCR Filter object if available.

Return Values

Only non-standard error values are listed:

Table 785

Name	Description
`UH_ERR_TSDMX_PCR_FILTER_NOT_AVAILABLE`	The allocation failed due to a lack of available PCR Filter objects.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— If `demux.nrPcrFilters < demux.maxNrPcrFilters`
  — Modify `demux.nrPcrFilters`
  — Modify `demux.pcrFilters`

Postcondition
— If `demux.nrPcrFilters' < demux.maxNrPcrFilters`
  — `*ppIPcrFilter` contains a pointer to the `uhITsDmxPcrFilter` interface of the allocated PCR Filter object `pcrFilter`.
  — `pcrFilter` belongs to the set of PCR Filters `demux.pcrFilters` associated with the `TsDemultiplexer`.
  — `demux.nrPcrFilters == demux.nrPcrFilters' + 1`
  — `0x0000 ≤ pcrFilter.pid ≤ 0x1FFF`
  — `pcrFilter.started == False`
— else `*ppIPcrFilter == Null`

8.1.13.5.2.4 AllocateStream

Signature
```
uhErrorCode_t AllocateStream (
    [in] uhTsDmx_StreamDataType_t streamDataType,
    [out] uhITsDmxStream           **ppIStream  );
```

Qualifiers
— synchronous

— thread-safe

Description
Allocates a Client Stream object and returns a pointer to the `uhITsDmxStream` interface of the allocated Client Stream object.

Parameters

Table 786

Name	Description
`streamDataType`	The data type of the Client Stream object to allocate.
`ppIStream`	Output parameter returning a pointer to the `uhITsDmxStream` interface of the allocated Client Stream object.

Return Values
Standard.

Precondition
`streamDataType != uhTsDmx_InternalFormat`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action

— Modify `demux.nrClientStreams`

— Modify `demux.clientStreams`

Postcondition

— `ppIStream` contains a pointer to the `uhITsDmxStream` interface of the allocated Client Stream object `clientStream`.
— `clientStream` belongs to the set of Client Streams `demux.clientStreams` associated with the `TsDemultiplexer`.
— demux.nrClientStreams == demux.nrClientStreams′ + 1
— `clientStream`.streamObjectType == ClientStream
— `clientStream`.dataType == `streamDataType`
— `clientStream`.outputType == ClientMemory
— `clientStream`.nrPidFilters == 0
— `clientStream`.pidFilters == {}
— `clientStream`.nrTphFilters == 0
— `clientStream`.tphFilters == {}
— `clientStream`.nrTransportPackets == 0
— `clientStream`.bufferType == `undefined`
— `clientStream`.bufferSize == 0
— `clientStream`.bufferGranularity == 0
— `clientStream`.pStartAddressBuffer == `Null`
— `clientStream`.pEndAddressBuffer == `Null`

Remarks

— Only Client Stream objects can be allocated using this function. `Platform' Stream objects can be accessed via the `GetPinRelatedObject` function (see 6.1.5) of the `uhIPinObjects` interface.

8.1.13.5.2.5 AllocateDescrambler

Signature

```
uhErrorCode_t AllocateDescrambler (
    [in] uhTsDmx_DescramblerType_t descramblerType,
    [out] uhITsDmxDescrambler       ** ppIDescrambler );
```

Qualifiers

— synchronous

— thread-safe

Description

Allocates a Descrambler object and returns a pointer to the `uhITsDmxDescrambler` interface of the allocated `Descrambler` object.

Parameters

Table 787

Name	Description
`descramblerType`	The data type of the Descrambler object to allocate.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
ppIDescrambler	Output parameter returning a pointer to the uhITsDmxDescrambler interface of the allocated Descrambler object if available.

Return Values
Only non-standard error values are listed:

Table 788

Name	Description
UH_ERR_TSDMX_DESCR_NOT_AVAILABLE	The allocation failed due to a lack of available Descrambler objects.

Precondition
- descramblerType in demux.suppDescramblerTypes
- if demux.descramblerTypeRestriction == True
  - for each t in uhTsDmx_DescramblerType_t where t!= descramblerType demux.nrDescramblers[t] == 0

Action
- If demux.nrDescramblers < demux.maxNrDescramblers
  - Modify demux.nrDescramblers
  - Modify demux.descramblers

Postcondition
- If demux.nrDescramblers' < demux.maxNrDescramblers
  - *ppIDescrambler contains a pointer to the uhITsDmxDescrambler interface of the allocated Descrambler object descrambler.
  - descrambler belongs to the set of Descramblers demux.descramblers associated with the TsDemultiplexer.
  - demux.nrDescramblers[descramblerType] == demux.nrDescramblers[descramblerType]' + 1
  - descrambler.descramblerType == descramblerType
  - descrambler.cw[] == undefined
  - descrambler.cwType == undefined
  - descrambler.defaultCwUsed == False
- else *ppIDescrambler == Null

8.1.13.5.2.6 GetSyncLockStatus

Signature
```
uhErrorCode_t GetSyncLockStatus (
    [out] puhTsDmx_SyncStatus_t pSyncLock );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns whether the demultiplexer is currently synchronized with the input stream (i.e., it did encounter the synchronization byte on several consecutive transport packets) or not.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 789

Name	Description
pSyncLock	Output parameter returning whether the Transport Stream Demultiplexing logical component is currently synchronized with the input stream, i.e. whether it has previously encountered the synchronization byte on several consecutive transport packets of the incoming transport stream and is, currently, actively processing the incoming transport stream.

Return Values
Standard

Precondition
`True`

Action
None.

Postcondition
`*pSyncLock == demux.syncLock`

8.1.13.5.3 uhITsDmxNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a `TsDemultiplexer`. The Subscribe function of the `uhITsDmx` interface can be used to subscribe to these events.

Interface ID
`uuid( 00612F6D-1A38-44C8-A23B-69C5F305DB1B )`

8.1.13.5.3.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                  cookie,
    [in] uhTsDmx_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.13.5.3.2 OnSyncLost

Signature
```
[async] Void OnSyncLost ( [in] UInt32 cookie );
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_SyncLost`. It is called when the `TsDemultiplexer` did not find the synchronization byte on several consecutive transport packets of the incoming transport stream after previously being synchronized, or in the invalid lock status.

Parameters

Table 790

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.13.5.3.3 OnSyncLock

Signature
```
[async] Void OnSyncLock ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_SyncLock`. It is called when the `TsDemultiplexer` did encounter the synchronization byte on several consecutive transport packets of the incoming transport stream after previously being out of sync, or in the invalid lock status.

Parameters

Table 791

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.13.5.3.4 OnSyncInvalid

Signature
```
[async] Void OnSyncInvalid (
    [in] UInt32 cookie
);
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_SyncInvalid`. It is called when the sync lock status has become invalid.

Parameters

Table 792

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.13.5.3.5 OnInputOverflow

Signature
```
[async] Void OnInputOverflow ( [in] UInt32 cookie );
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_InputOverflow`. It is called when the `TsDemultiplexer` cannot become synchronized on the incoming transport stream due to its high bit rate.

Parameters

Table 793

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.13.5.4 uhITsDmxCapability

Qualifiers
None.

Description
This interface of a `TsDemultiplexer` provides facilities to retrieve the capability of filters and descramblers. For descramblers and each type of filter, the capabilities are the maximum number of filters and the number of currently available filters.

Interface ID
`uuid( 8301FA19-B25D-46DD-BAAC-7E86C53BA83E )`

8.1.13.5.4.1 GetSuppDescramblerTypes

Signature
```
uhErrorCode_t GetSuppDescramblerTypes (
    [out] puhTsDmx_DescramblerTypes_t pSuppDescramblerTypes );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Returns the descrambler types supported by the Transport Stream Demultiplexing logical component.

Parameters

Table 794

Name	Description
`pSuppDescramblerTypes`	Output parameter returning the descrambler types supported by the Transport Stream Demultiplexing logical component.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pSuppDescramblerTypes == demux.suppDescramblerTypes`

8.1.13.5.4.2 GetDescramblerTypeRestriction

Signature
```
uhErrorCode_t GetDescramblerTypeRestriction (
    [out] pBool pRestriction );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether at any point in time all of the allocated descramblers have to be of the same type.

Parameters

Table 795

Name	Description
`pRestriction`	Output parameter returning whether at any point in time all of the allocated descramblers have to be of the same type. It is `True` when at any point in time all of the allocated descramblers have to be of the same type. It is `False`, otherwise.

Return Values
Standard

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
`*pRestriction == demux.descramblerTypeRestriction`

8.1.13.5.4.3 GetSuppBufferTypes

Signature
```
uhErrorCode_t GetSuppBufferTypes (
    [out] puhTsDmx_BufferTypes_t pSuppBufferTypes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the set of buffer types supported by the platform.

Parameters

Table 796

Name	Description
`pSuppBufferTypes`	Output parameter returning the set of buffer types supported by the platform

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pSuppBufferTypes == demux.suppBufferTypes`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.4.4 GetPidRestriction

Signature
```
uhErrorCode_t GetPidRestriction (
    [out] pBool pRestriction );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether there is a restriction on using at any point in time the same PID value on more than one started PID/TPH filter.

Parameters

Table 797

Name	Description
`pRestriction`	Output parameter returning whether there is a restriction on using at any point in time the same PID value on more than one started PID/TPH filter. It is `True` when the restriction applies: using the same PID value on more than one started PID/TPH filter is not supported. It is `False` when the restriction does not apply: using the same PID value on more than one started PID/TPH filter is supported.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pRestriction == demux.pidRestriction`

8.1.13.5.4.5 GetMaxNrPidFilters

Signature
```
uhErrorCode_t GetMaxNrPidFilters ( [out] pUInt32 pMaxNrPidFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum number of PID Filters supported by the Transport Stream Demultiplexing logical component.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 798

Name	Description
pMaxNrPidFilters	Output parameter returning the maximum number of PID Filters supported by the Transport Stream Demultiplexing logical component.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pMaxNrPidFilters == demux.maxNrPidFilters

8.1.13.5.4.6 GetAvailableNrPidFilters

Signature
```
uhErrorCode_t GetAvailableNrPidFilters (
    [out] pUInt32 pAvailableNrPidFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of available PID Filters.

Parameters

Table 799

Name	Description
pAvailableNrPidFilters	Output parameter returning the number of available PID Filters.

Return Values
Standard.

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— *pAvailableNrPidFilters == demux.maxNrPidFilters-demux.nrPidFilters

8.1.13.5.4.7 GetMaxNrTphFilters

Signature
```
uhErrorCode_t GetMaxNrTphFilters ( [out] pUInt32 pMaxNrTphFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum number of TPH Filters supported by the Transport Stream Demultiplexing logical component.

Parameters

Table 800

Name	Description
pMaxNrTphFilters	Output parameter returning the maximum number of TPH Filters supported by the Transport Stream Demultiplexing logical component.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pMaxNrTphFilters == demux.maxNrTphFilters

8.1.13.5.4.8 GetAvailableNrTphFilters

Signature
```
uhErrorCode_t GetAvailableNrTphFilters (
    [out] pUInt32 pAvailableNrTphFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of available TPH Filters.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 801

Name	Description
pAvailableNrTphFilters	Output parameter returning the number of available TPH Filters.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pAvailableNrTphFilters == demux.maxNrTphFilters-demux.nrTphFilters

8.1.13.5.4.9 GetMaxNrSectionFilters

Signature
```
uhErrorCode_t GetMaxNrSectionFilters ( [out] pUInt32 pMaxNrSectionFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum number of Section Filters supported by the Transport Stream Demultiplexing logical component.

Parameters

Table 802

Name	Description
pMaxNrSectionFilters	Output parameter returning the maximum number of Section Filters supported by the Transport Stream Demultiplexing logical component.

Return Values
Standard.

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `*pMaxNrSectionFilters == demux. maxNrSectionFilters`

8.1.13.5.4.10 GetAvailableNrSectionFilters

Signature
```
uhErrorCode_t GetAvailableNrSectionFilters (
    [out] pUInt32 pAvailableNrSectionFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of available Section Filters.

Parameters

Table 803

Name	Description
`pAvailableNrSectionFilters`	Output parameter returning the number of available Section Filters.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pAvailableNrSectionFilters == demux.maxNrSectionFilters-demux.nrSectionFilters`

8.1.13.5.4.11 GetMaxNrPesFilters

Signature
```
uhErrorCode_t GetMaxNrPesFilters ( [out] pUInt32 pMaxNrPesFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum number of PES Filters supported by the Transport Stream Demultiplexing logical component.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 804

Name	Description
pMaxNrPesFilters	Output parameter returning the maximum number of PES Filters supported by the Transport Stream Demultiplexing logical component.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pMaxNrPesFilters == demux.maxNrPesFilters

8.1.13.5.4.12 GetAvailableNrPesFilters

Signature
```
uhErrorCode_t GetAvailableNrPesFilters (
    [out] pUInt32 pAvailableNrPesFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of available PES Filters.

Parameters

Table 805

Name	Description
pAvailableNrPesFilters	Output parameter returning the number of available PES Filters.

Return Values
Standard.

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `*pAvailableNrPesFilters == demux.maxNrPesFilters-demux.nrPesFilters`

8.1.13.5.4.13 GetMaxNrPcrFilters

Signature
```
uhErrorCode_t GetMaxNrPcrFilters ( [out] pUInt32 pMaxNrPcrFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum number of PCR Filters supported by the Transport Stream Demultiplexing logical component.

Parameters

Table 806

Name	Description
`pMaxNrPcrFilters`	Output parameter returning the maximum number of PCR Filters supported by the Transport Stream Demultiplexing logical component.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMaxNrPcrFilters == demux.maxNrPcrFilters`

8.1.13.5.4.14 GetAvailableNrPcrFilters

Signature
```
uhErrorCode_t GetAvailableNrPcrFilters (
    [out] pUInt32 pAvailableNrPcrFilters );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of available PCR Filters.

-`,,```,,,,````-`-,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 807

Name	Description
pAvailableNrPcrFilters	Output parameter returning the number of available PCR Filters.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pAvailableNrPcrFilters == demux.maxNrPcrFilters-demux.nrPcrFilters

8.1.13.5.4.15 GetMaxNrDescramblers

Signature

```
uhErrorCode_t GetMaxNrDescramblers (
    [in] uhTsDmx_DescramblerType_t descramblerType,
    [out] pUInt32                  pMaxNrDescramblers );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum number of Descramblers of type descramblerType supported by the Transport Stream Demultiplexing logical component.

Parameters

Table 808

Name	Description
descramblerType	The descrambler type.
pMaxNrDescramblers	Output parameter returning the maximum number of Descramblers of type descramblerType supported by the Transport Stream Demultiplexing logical component.

Return Values
Standard.

Precondition
True

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— `*pMaxNrDescramblers == demux.maxNrDescramblers[descramblerType]`

8.1.13.5.4.16 GetAvailableNrDescramblers

Signature
```
uhErrorCode_t GetAvailableNrDescramblers (
    [in] uhTsDmx_DescramblerType_t descramblerType,
    [out] pUInt32                  pAvailableNrDescramblers );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of available `Descramblers` of type descramblerType.

Parameters

Table 809

Name	Description
`descramblerType`	The descrambler type.
`pAvailableNrDescramblers`	Output parameter returning the number of available `Descramblers` of type descramblerType.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pAvailableNrDescramblers == demux.maxNrDescramblers[descramblerType]-demux.nrDescramblers[descramblerType]`

8.1.13.5.5 uhITsDmxTimeStampInput

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface of a `TsDemultiplexer` provides a means to control the time stamp mode in which the `TsDemultiplexer works.`

Interface ID
`uuid( A731D3CC-7AF2-4CF6-B091-660E09A3A837 )`

8.1.13.5.5.1 SetTimeStampMode

Signature
```
uhErrorCode_t SetTimeStampMode (
    [in] uhTsDmx_TimeStampMode_t timeStampMode );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the time stamp mode in which the `TsDemultiplexer works.`

Parameters

Table 810

Name	Description
`timeStampMode`	Time stamp mode in which the `TsDemultiplexer works.`

Return Values
Only non-standard error values are listed:

Table 811

Name	Description
`UH_ERR_TSDMX_TIME_STAMP_MODE_NOT_SET`	The update of the time stamp mode of the TS Demux did not occur: setting the time stamp mode of the TS Demux can be effectively performed only when the TS Demux is not actively processing an input transport stream, i.e. when `demux.syncLock == uhTsDmx_SyncStatusInvalid`, or when `demux.syncLock == uhTsDmx_SyncStatusLost`.

Precondition
`True`

Action
— Modify `demux.timeStampMode`

Postcondition
`demux.timeStampMode == timeStampMode`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.5.2 GetTimeStampMode

Signature
```
uhErrorCode_t GetTimeStampMode (
    [out] puhTsDmx_TimeStampMode_t pTimeStampMode );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current time stamp mode in which the `TsDemultiplexer works.`

Parameters

Table 812

Name	Description
`pTimeStampMode`	Output parameter returning the current time stamp mode in which the `TsDemultiplexer works.`

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pTimeStampMode == demux.timeStampMode`

8.1.13.5.6 uhITsDmxDescrambler

Qualifiers
None.

Description
This interface of a `Descrambler` provides facilities to set control words to be used for descrambling.

Interface ID
`uuid( D365FEA1-BCB2-4d36-905F-156691B179C9 )`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.6.1 SetControlWord

Signature

```
uhErrorCode_t SetControlWord (
    [in] uhTsDmx_CwType_t            cwType,
    [in] UInt32                      size,
    [in, size_is(size)] const pUInt8 pCw );
```

Qualifiers

— synchronous

— single-threaded

Description

Sets a control word.

Parameters

Table 813

Name	Description
`cwType`	The type of the control word.
`size`	The size in bytes of the control word.
`pCw`	Pointer to a control word with size bytes of valid data.

Return Values

Standard.

Precondition

— Let `descrambler` be the `Descrambler` object on which this interface operates.
— `size ==` demux.cwDepth`[descrambler.`descramblerType`]`
— `descrambler.`defaultCwUsed `== False`

Action

— Modify `descrambler.`cwType
— Modify `descrambler.cw`

Postcondition

— `descrambler.cwType == cwType`
— For each `i` in `{0 .. size-1}`
  — `descrambler.cw[i] == pCw[i]`

8.1.13.5.6.2 UseDefaultControlWord

Signature

```
uhErrorCode_t UseDefaultControlWord ( [in] Bool useDefaultCw );
```

Qualifiers

— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— single-threaded

Description
Sets whether to use the default control word for descrambling.

Parameters

Table 814

Name	Description
`useDefaultCw`	Indicates whether to use the default control word for descrambling. It is `True` when the default control word should be used. It is `False` when the default control word should not be used.

Return Values
Standard.

Precondition
`True`

Action
— Let `descrambler` be the `Descrambler` object on which this interface operates.
— Modify `descrambler.defaultCwUsed`

Postcondition
— `descrambler.defaultCwUsed == useDefaultCw`

8.1.13.5.6.3 GetControlWordDepth

Signature
```
uhErrorCode_t GetControlWordDepth (
    [out] pUInt32                  pControlWordDepth );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the control word depth - in bytes - of the `Descrambler` object. The control word depth is the size of a control word in bytes.

Parameters

Table 815

Name	Description
`pControlWordDepth`	Output parameter returning the filter depth - in bytes.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
— Let `descrambler` be the `Descrambler` object on which this interface operates.
— `*pControlWordDepth ==` demux.cwDepth`[descrambler.`descramblerType`]`

8.1.13.5.7 uhITsDmxStream

Qualifiers
None.

Description
This interface of a Stream object provides facilities to allocate PID Filters and TPH Filters and to retrieve the number of transport packets that passed the PID Filter condition set on the associated PID or TPH Filter object(s).

Interface ID
`uuid( 0A1FB286-9D06-4a4b-A124-B6D58AA352F4 )`

8.1.13.5.7.1 AllocatePidFilter

Signature
```
uhErrorCode_t AllocatePidFilter ( [out] uhITsDmxPidFilter **ppIPidFilter );
```

Qualifiers
— synchronous

— thread-safe

Description
Allocates a PID Filter object and returns a pointer to the `uhITsDmxPidFilter` interface of the allocated PID Filter object.

Parameters

Table 816

Name	Description
`ppIPidFilter`	Output parameter returning a pointer to the `uhITsDmxPidFilter` interface of the allocated PID Filter object if available.

Return Values
Only non-standard error values are listed:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 817

Name	Description
UH_ERR_TSDMX_PID_FILTER_NOT_AVAILABLE	The allocation failed due to a lack of available PID Filter objects.

Precondition

— The Stream object does not have an association with a PID or a TPH Filter object in case the data / output type of this `Stream` object differs from one of the following pairs (which are used for partial transport stream generation):

— `(uhTsDmx_Tp, ClientMemory)`,

— `(uhTsDmx_InternalFormat, Remultiplexer)` (see Figure 88).

Action

— Let `stream` be the `Stream` object on which this interface operates.

— If `demux.nrPidFilters < demux.maxNrPidFilters`

  — Modify `demux.nrPidFilters`

  — Modify `stream.nrPidFilters`

  — Modify `stream.pidFilters`

Postcondition

— If `demux.nrPidFilters' < demux.maxNrPidFilters`

  — `*ppIPidFilter` contains a pointer to the `uhITsDmxPidFilter` interface of the allocated PID Filter object `pidFilter`.

  — `pidFilter` belongs to the set of PID Filters `stream.pidFilters` associated with the Stream object `stream`.

  — `demux.nrPidFilters == demux.nrPidFilters' + 1`

  — `stream.nrPidFilters == stream.nrPidFilters' + 1`

  — `0x0000 ≤ pidFilter.pid ≤ 0x1FFF`

  — `pidFilter.started == False`

  — `pidFilter.nrSectionFilters == 0`

  — `pidFilter.sectionFilters == {}`

  — `pidFilter.nrPesFilters == 0`

  — `pidFilter.pesFilters == {}`

  — `pidFilter.pIPesNotifier == Null`

  — `pidFilter.scramblingStatus == uhTsDmx_ScramblingUnknown`

  — `pidFilter.pIDescrambler == Null`

— else `*ppIPidFilter == Null`

8.1.13.5.7.2 AllocateTphFilter

Signature

```
uhErrorCode_t AllocateTphFilter ( [out] uhITsDmxPidFilter **ppITphFilter );
```

Qualifiers

— synchronous

— thread-safe

Description

Allocates a TPH Filter object and returns a pointer to the `uhITsDmxPidFilter` interface of the allocated TPH Filter object.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 818

Name	Description
ppITphFilter	Pointer to the uhITsDmxPidFilter interface of the allocated TPH Filter object if available.

Return Values
Only non-standard error values are listed:

Table 819

Name	Description
UH_ERR_TSDMX_TPH_FILTER_NOT_AVAILABLE	The allocation failed due to a lack of available TPH Filter objects.

Precondition
- The Stream object does not have an association with a PID or a TPH Filter object in case the data / output type of this Stream object differs from one of the following pairs (which are used for partial transport stream generation):
- (uhTsDmx_Tp, ClientMemory),
- (uhTsDmx_InternalFormat, Remultiplexer) (see Figure 88).

Action
- Let stream be the Stream object on which this interface operates.
- If demux.nrTphFilters < demux.maxNrTphFilters
  - Modify demux.nrTphFilters
  - Modify stream.nrTphFilters
  - Modify stream.tphFilters

Postcondition
- If demux.nrTphFilters' < demux.maxNrTphFilters
  - *ppITphFilter contains a pointer to the uhITsDmxPidFilter interface of the allocated TPH Filter object tphFilter.
  - tphFilter belongs to the set of TPH Filters stream.tphFilters associated with the Stream object stream.
  - demux.nrTphFilters == demux.nrTphFilters' + 1
  - stream.nrPidFilters == stream.nrPidFilters' + 1
  - 0x0000 ≤ tphFilter.pid ≤ 0x1FFF
  - tphFilter.started == False
  - tphFilter.nrPesFilters == 0
  - tphFilter.pesFilters == {}
  - tphFilter.pIPesNotifier == Null
  - tphFilter.scramblingStatus == uhTsDmx_ScramblingUnknown
  - tphFilter.pIDescrambler == Null
  - tphFilter.pMatchValue[] == undefined
  - tphFilter.pMaskValue[] == undefined
- else *ppITphFilter == Null

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.7.3 GetNrTransportPackets

Signature
```
uhErrorCode_t GetNrTransportPackets ( [out] pUInt32 pNrTransportPackets );
```

Qualifiers
- synchronous
- thread-safe

Description
For non-SPTS generation, it returns the number of transport packets that passed the PID Filter condition set on a PID or TPH Filter object associated to the Stream object since the last time this function was called.

For SPTS generation, this function returns the number of transport packets that passed the union of the PID Filter conditions of all PID and TPH Filter object associated to the Stream object since the last time this function was called.

Parameters

Table 820

Name	Description
`pNrTransportPackets`	Output parameter returning the number of transport packets that passed the PID Filter condition set on the PID or TPH Filter since the last time this function was called.

Return Values
Standard.

Precondition
- Let `stream` be the `Stream` object on which this interface operates.
- `stream.dataType == uhTsDmx_InternalFormat`

Action
- `dummyNrTp == stream.nrTransportPackets`
- Modify `stream.nrTransportPackets`

Postcondition
- `stream.nrTransportPackets == 0`
- `*pNrTransportPackets == dummyNrTp`

Remarks
- If the number of transport packets - received after the last time this function was called - exceeds the maximum number that a `UInt32` can hold, it will wrap around and restart from 0.

8.1.13.5.8 uhITsDmxStreamBuffer

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface of a Client Stream object provides facilities to retrieve and set the configuration of the buffer associated with the Client Stream object. It also contains a function to retrieve the pointers to the beginning and the end of the buffer and a function to release a lock on data that has been notified.

It also provides a `Subscribe` and an `Unsubscribe` function for notifications of events related to the buffer of the Client Stream object.

Interface ID
`uuid( D016F06A-1B4B-4377-88B8-A021922C7736 )`

8.1.13.5.8.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhITsDmxStreamBufferNtf             *pINotify,
    [in] UInt32                              cookie,
    [in] uhTsDmx_StreamBufferNotifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.13.5.8.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhITsDmxStreamBufferNtf             *pINotify,
    [in] UInt32                              cookie,
    [in] uhTsDmx_StreamBufferNotifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.13.5.8.3 GetBufferConfig

Signature
```
uhErrorCode_t GetBufferConfig (
    [out] puhTsDmx_BufferType_t pBufferType,
    [out] pUInt32               pBufferSize,
    [out] pUInt32               pGranularity );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the configuration of the buffer associated with the Client Stream object.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 821

Name	Description
pBufferType	Output parameter returning the type of the buffer associated with the Client Stream object.
pBufferSize	Output parameter returning the size - in bytes - of the buffer associated with the Client Stream object.
pGranularity	Output parameter returning the granularity - in bytes - of the buffer associated with the Client Stream object.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— Let stream be the ClientStream object on which this interface operates.
— *pBufferType == stream.bufferType
— *pBufferSize == stream.bufferSize
— *pGranularity == stream.bufferGranularity

Remarks
When *pGranularity == 0, only complete PES packets are notified through the OnCompletePacketArrived notification of the uhITsDmxPesNotifierNtf notification interface, no (partial) PES packets will be notified through the OnPartialPacketArrived notification.

8.1.13.5.8.4 SetBufferConfig

Signature

```
uhErrorCode_t SetBufferConfig (
    [in] uhTsDmx_BufferType_t bufferType,
    [in] UInt32               requestedBufferSize,
    [in] UInt32               requestedGranularity,
    [out] pUInt32             pActualBufferSize,
    [out] pUInt32             pActualGranularity );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the configuration of the buffer associated with the Client Stream object.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 822

Name	Description
`bufferType`	The buffer type.
`requestedBufferSize`	The requested size - in bytes - of the buffer associated with the `Client Stream` object.
`requestedGranularity`	The requested granularity - in bytes - of the buffer associated with the `Client Stream` object. If partial packet notification is supported, it is a lower bound for the returned granularity `*pActualGranularity`.
`pActualBufferSize`	Output parameter returning the actual size - in bytes - of the buffer associated with the `Client Stream` object.
`pActualGranularity`	Output parameter returning the actual granularity - in bytes - of the buffer associated with the `Client Stream` object.

Return Values
Standard.

Precondition
- Only buffers that are associated to a Client Stream object of data type `uhTsDmx_Pes` or `uhTsDmx_Tp` can have a granularity different from 0. For all others, the granularity is 0.
- The minimum granularity for transport packets either equals the transport packet size or equals 0. The returned granularity will be always a multiple of the transport packet size or 0.
- `bufferType` in `stream.suppBufferTypes`

Action
- Let `stream` be the `ClientStream` object on which this interface operates.
- Modify `stream.bufferType`
- Modify `stream.bufferSize`
- Modify `stream.bufferGranularity`

Postcondition
- `stream.bufferType == bufferType`
- `stream.bufferSize == a number greater or equal to 0 determined by the platform`
- If partial packet notification is supported
  - `stream.bufferGranularity` ≥ `requestedGranularity`
- else
  - `stream.bufferGranularity == 0`
- `*pActualBufferSize == stream.bufferSize`
- `*pActualGranularity == stream.bufferGranularity`

Remarks
- For PES data, the granularity specifies the size of a partial PES packet that will be notified.
- If partial packet notification is supported, then the requested granularity `requestedGranularity` is a lower bound for the returned granularity; if partial packet notification is not supported, then the returned granularity will be 0.
- If a client does not want any granularity for data of type uhTsDmx_Pes or uhTsDmx_Tp, the `requestedGranularity` should be 0.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— When `*pActualGranularity == 0`, only complete PES packets are notified through the `OnCompletePacketArrived` notification of the `uhITsDmxPesNotifierNtf` notification interface, no (partial) PES packets will be notified through the `OnPartialPacketArrived` notification.

8.1.13.5.8.5 Free

Signature

```
uhErrorCode_t Free ( [in] UInt32 dataCookie );
```

Qualifiers

— synchronous

— single-threaded

Description

Releases a lock on notified data in the buffer associated with the Client Stream object. Only when all objects that have been notified for a certain data unit have released their lock on this data unit, this data unit will be freed.

Parameters

Table 823

Name	Description
`dataCookie`	Value passed via a notification function of a notify interface when a complete data unit has been filtered and written to a buffer.

Return Values

Standard.

Precondition

— dataCookie identifies a previously filtered, notified, and not yet released data unit that is present in the buffer associated with the Client Stream.

Action

None.

Postcondition

— A lock on the data identified by dataCookie has been released.

Remarks

— If a client unlocks a partial PES packet using this function, it implicitly unlocks all partial PES packets that are part of the same PES packet and that are notified and written to the buffer associated with the Client Stream object **prior** to this partial PES packet. The lock status of any PES packet data that are part of the same PES packet and that are notified and written to the buffer associated with the Client Stream object **after** to this partial PES packet remains unaffected.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.9 uhITsDmxStreamBufferNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a Client Stream object. The `Subscribe` function of the `uhITsDmxStreamBuffer` can be used to subscribe to these events.

Interface ID
`uuid( E7E489A8-2EB9-49f7-9FCE-1A5DC29ECB91 )`

8.1.13.5.9.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
  [in] UInt32                          cookie,
  [in] uhTsDmx_StreamBufferNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.13.5.9.2 OnBufferFull

Signature
```
[async] Void OnBufferFull ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_StreamBufferFull`. It is called if the Client Stream object does not have enough free space in its associated buffer for storing newly received data.

Parameters

Table 824

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
— In case a buffer does not hold enough free space to store data, this data is discarded and the filter process is paused. Furthermore, data which arrives after the pausing of the filter process will be discarded as well, even if this data is smaller in size and might fit in the remaining free buffer space. If the filter process is paused due to lack of free buffer space, it will be resumed in case a client releases the lock on some data and frees buffer space in a way that there is enough consecutive space from the position in the buffer where the last data has been written to store new data.

8.1.13.5.10 uhITsDmxTimeStampOutput

Qualifiers
None.

Description
This interface of a `PlatformStream` provides functionality to enable or disable the addition of time stamps to the outputted (platform) stream.

Interface ID
`uuid( C3CF2E71-EF2D-417D-BF10-BEE6A30CB4DC )`

8.1.13.5.10.1 EnableTimeStamps

Signature
```
uhErrorCode_t EnableTimeStamps (
    [in] Bool enable );
```

Qualifiers
— synchronous

— single-threaded

Description
This function is used by the client to indicate that the addition of time stamps to the outputted (platform) stream must be enabled or disabled.

Parameters

Table 825

Name	Description
`enable`	Indication on whether the addition of time stamps to the outputted (platform) stream must be enabled or disabled. It is `True` if the addition of time stamps to the outputted (platform) stream must be enabled. It is `False` otherwise.

--`,,`,,`,,

Return Values
Standard.

Precondition
— Let `stream` be the `PlatformStream` object on which this interface operates.
— `for each pidFilter` belonging to the set of PID Filters `stream.pidFilters` associated with `stream`:
  — `pidFilter.started == False`

Action
— Modify `stream.enableTimeStamps`

Postcondition
`stream.enableTimeStamps == enable`

8.1.13.5.10.2 GetTimeStampsEnabled

Signature
```
uhErrorCode_t GetTimeStampsEnabled (
    [out] pBool pEnable );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the enable/disable setting set by the client on the addition of time stamps to the outputted (platform) stream.

Parameters

Table 826

Name	Description
`pEnable`	Output parameter returning wether the addition of time stamps to the outputted (platform) stream is enabled or disabled. It is `True` if the addition of time stamps to the outputted (platform) stream is enabled. It is `False` otherwise.

Return Values
Standard.

Precondition
— Let `stream` be the `PlatformStream` object on which this interface operates.

Action
None.

Postcondition
`*pEnable == stream.enableTimeStamps`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.11 uhITsDmxPcrFilter

Qualifiers
None.

Description
This interface of a PCR Filter provides facilities to start and to stop the PCR Filter and to retrieve and to set a PID. It also provides a `Subscribe` and `Unsubscribe` function on notifications of events that occur in the PCR Filter.

Interface ID
`uuid( 76E4B579-432F-43E2-8666-E372F36B9829 )`

8.1.13.5.11.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhITsDmxPcrFilterNtf       *pINotify,
    [in] UInt32                     cookie,
    [in] uhTsDmx_PcrNotifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.13.5.11.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhITsDmxPcrFilterNtf       *pINotify,
    [in] UInt32                     cookie,
    [in] uhTsDmx_PcrNotifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.13.5.11.3 Start

Signature
```
uhErrorCode_t Start ( );
```

Qualifiers
— synchronous

— single-threaded

Description
Starts the PCR Filter object on which this interface operates.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters
None.

Return Values
Standard.

Precondition
`True`

Action
— Let `pcrFilter` be the `PcrFilter` object on which this interface operates.

— Modify `pcrFilter.started`

Postcondition
— `pcrFilter.started == True`

8.1.13.5.11.4 Stop

Signature
```
uhErrorCode_t Stop ( );
```

Qualifiers
— asynchronous

— single-threaded

Description
Sends a stop request to the PCR Filter object on which this interface operates.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Asynchronous **Action**
— Let `pcrFilter` be the `PcrFilter` object on which this interface operates.

— Modify `pcrFilter.started`

— Notify `OnStopped(*)`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Asynchronous Postcondition
— `pcrFilter.started == False`

8.1.13.5.11.5 GetPid

Signature
`uhErrorCode_t GetPid ( [out] pUInt16 pPid );`

Qualifiers
— synchronous

— thread-safe

Description
Returns the current PID of the PCR Filter.

Parameters

Table 827

Name	Description
`pPid`	Output parameter returning the current PID.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `pcrFilter` be the `PcrFilter` object on which this interface operates.

— `*pPid == pcrFilter.pid`

8.1.13.5.11.6 SetPid

Signature
`uhErrorCode_t SetPid ( [in] UInt16 pid );`

Qualifiers
— synchronous

— single-threaded

Description
Sets the PID of the PCR Filter.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 828

Name	Description
pid	The PID.

Return Values
Standard.

Precondition
— `0x0000 <= pid <= 0x1FFF`
— Let `pcrFilter` be the `PcrFilter` object on which this interface operates.
— `pcrFilter.started == False`

Action
— Modify `pcrFilter.pid`

Postcondition
`pcrFilter.pid == pid`

8.1.13.5.12 uhITsDmxPcrFilterNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a PCR Filter. The `Subscribe` function of the `uhITsDmxPcrFilter` interface can be used to subscribe to these events.

Interface ID
`uuid( 9FD894AC-B68F-40bb-AA82-9EB1C922CB94 )`

8.1.13.5.12.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                    cookie,
    [in] uhTsDmx_PcrNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.13.5.12.2 OnStopped

Signature
```
[async] Void OnStopped ( [in] UInt32 cookie );
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_PcrFilterStopped`. It is called in response to a `Stop` call to indicate that a filter object has stopped.

Parameters

Table 829

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
Once received, the client will not receive any other event from the filter object on which the corresponding control interface operates before being started again, except for `uhTsDmx_PcrFilterStopped` when an already stopped filter object is stopped again.

8.1.13.5.13 uhITsDmxPidFilter

Qualifiers
None.

Description
This interface of a PID or TPH Filter provides facilities to start and to stop the filter and to retrieve and to set a PID. It also provides a `Subscribe` and `Unsubscribe` function on notifications of events that occur in the filter.

This interface also provides facilities to allocate Section Filters, PES Filters, and PES Notifiers.

Interface ID
`uuid( 42D98FE1-AF29-4C08-802A-EA2577266D3A )`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.13.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhITsDmxPidFilterNtf       *pINotify,
    [in] UInt32                     cookie,
    [in] uhTsDmx_PidNotifications_t notifs );
```

Qualifiers

— subscribe-function

8.1.13.5.13.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhITsDmxPidFilterNtf       *pINotify,
    [in] UInt32                     cookie,
    [in] uhTsDmx_PidNotifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.13.5.13.3 Start

Signature

```
uhErrorCode_t Start ( );
```

Qualifiers

— synchronous

— single-threaded

Description

Starts the PID or TPH Filter object on which this interface operates.
If the Stream object - that is associated with the PID or TPH Filter object on which this interface operates - has data type `uhTsDmx_MpegSection` and output type `ClientMemory`, then if no Section Filter is associated with the PID or TPH Filter object or no associated Section Filter is started, it filters nothing.

Parameters

None.

Return Values

Only non-standard error values are listed:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 830

Name	Description
UH_ERR_TSDMX_FILTER_NOT_STARTED	Starting the PID or TPH Filter failed: the restriction on the use of the same PID value on more than one started PID/TPH filter applies (i.e. demux.pidRestriction == True) and there is another PID or TPH Filter - associated with another Stream that has another data type - that is already filtering data (i.e. that is in the `started' state) on the same PID.

Precondition
True

Action
— Let filter be the PidFilter or TphFilter object on which this interface operates.
— Modify filter.started

Postcondition
— filter.started == True

8.1.13.5.13.4 Stop

Signature
```
uhErrorCode_t Stop ( );
```

Qualifiers
— asynchronous

— single-threaded

Description
Sends a stop request to the PID or TPH Filter object on which this interface operates.

If the Stream object - associated with the PID or TPH Filter object on which this interface operates - is a Platform Stream (i.e. stream.StreamType == PlatformStream), then the data associated with this PID or TPH filter in the buffer used by the Stream object is flushed.

Parameters
None.

Return Values
Standard.

Precondition
True

Asynchronous Action
— Let pidTphFilter be the PidFilter or TphFilter object on which this interface operates.
— Modify pidTphFilter.started
— Notify OnStopped(*)

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Asynchronous Postcondition
— `pidTphFilter.started == False`

8.1.13.5.13.5 GetPid

Signature
```
uhErrorCode_t GetPid ( [out] pUInt16 pPid );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current PID of the PID or TPH Filter on which this interface operates.

Parameters

Table 831

Name	Description
`pPid`	Output parameter returning the current PID.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let filter be the `PidFilter` or `TphFilter` object on which this interface operates.
— `*pPid == filter.pid`

8.1.13.5.13.6 SetPid

Signature
```
uhErrorCode_t SetPid ( [in] UInt16 pid );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the PID of the PID or TPH Filter on which this interface operates.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 832

Name	Description
`pid`	The PID.

Return Values
Standard.

Precondition
— `0x0000 <= pid <= 0x1FFF`

— Let `filter` be the `PidFilter` or `TphFilter` object on which this interface operates.

`filter.started == False`

Action
— Modify `filter.pid`

Postcondition
— `filter.pid == pid`

8.1.13.5.13.7 AllocateSectionFilter

Signature
```
uhErrorCode_t AllocateSectionFilter (
    [out] uhITsDmxSectionFilter **ppISectionFilter );
```

Qualifiers
— synchronous

— thread-safe

Description
Allocates a Section Filter object and returns a pointer to the `uhITsDmxSectionFilter` interface of the allocated Section Filter.

Parameters

Table 833

Name	Description
`ppISectionFilter`	Output parameter returning a pointer to the `uhITsDmxSectionFilter` interface of the allocated Section Filter object.

Return Values
Only non-standard error values are listed:

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 834

Name	Description
UH_ERR_TSDMX_SCT_FILTER_NOT_AVAILABLE	The allocation failed due to a lack of available Section Filter objects.

Precondition
- The Stream object - that is associated with the PID filter object on which this interface operates - has data type uhTsDmx_MpegSection and output type ClientMemory.
- The filter object on which this interface operates is a PID Filter.
- Let filter be the PidFilter object on which this interface operates.
- filter.nrPesFilters == 0
- filter.pIPesNotifier == Null

Action
- If demux.nrSectionFilters < demux.maxNrSectionFilters
  - Modify demux.nrSectionFilters
  - Modify filter.nrSectionFilters
  - Modify filter. sectionFilters

Postcondition
- If demux.nrSectionFilters' < demux.maxNrSectionFilters
  - *ppISectionFilter contains a pointer to the uhITsDmxSectionFilter interface of the allocated Section Filter object sectionFilter.
  - demux.nrSectionFilters == demux.nrSectionFilters' + 1
  - sectionFilter belongs to the set of Section Filters filter.sectionFilters associated with the PID Filter object on which this interface operates.
  - filter.nrSectionFilters == filter.nrSectionFilters' + 1
  - sectionFilter.started == False
  - sectionFilter.filterMode == uhTsDmx_Continuous
  - sectionFilter.pModeValue == undefined
  - sectionFilter.pMatchValue == undefined
  - sectionFilter.pMaskValue == undefined
  - sectionFilter.crcMode == uhTsDmx_DontCheckCrc
  - sectionFilter.crc == undefined
- else *ppISectionFilter == Null

8.1.13.5.13.8 AllocatePesFilter

Signature
```
uhErrorCode_t AllocatePesFilter ( [out] uhITsDmxPesFilter **ppIPesFilter );
```

Qualifiers
- synchronous
- thread-safe

Description
Allocates a PES Filter object and returns a pointer to the uhITsDmxPesFilter interface of the allocated PES Filter object.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 835

Name	Description
ppIPesFilter	Output parameter returning a pointer to the uhITsDmxPesFilter interface of the allocated PES Filter object.

Return Values
Only non-standard error values are listed:

Table 836

Name	Description
UH_ERR_TSDMX_PES_FILTER_NOT_AVAILABLE	The allocation failed due to a lack of available PES Filter objects.

Precondition
- The Stream object - that is associated with the PID or TPH Filter object on which this interface operates - has:
  (data type uhTsDmx_Pes and output type ClientMemory) OR
  (data type uhTsDmx_InternalFormat and output type (VideoDecoder or AudioDecoder or TeletextDecoder or SubtitleDecoder)).
- Let filter be the PidFilter or TphFilter object on which this interface operates.
- filter.pIPesNotifier == Null
- If the filter object on which this interface operates is a PID Filter
  - filter.nrSectionFilters == 0

Action
- If demux.nrPesFilters < demux.maxNrPesFilters
  - Modify demux.nrPesFilters
  - Modify filter.nrPesFilters
  - Modify filter.pesFilters

Postcondition
- If demux.nrPesFilters' < demux.maxNrPesFilters
  - *ppIPesFilter contains a pointer to the uhITsDmxPesFilter interface of the allocated PES Filter object pesFilter.
  - demux.nrPesFilters == demux.nrPesFilters' + 1
  - pesFilter belongs to the set of PES Filters filter.pesFilters associated with the PID or TPH Filter object on which this interface operates.
  - filter.nrPesFilters == filter.nrPesFilters' + 1
  - pesFilter.started == False
  - pesFilter.filterMode == uhTsDmx_Continuous
  - pesFilter.pMatchValue == undefined
  - pesFilter.pMaskValue == undefined
  - pesFilter.minDts.timeStampHigh == 0
  - pesFilter.minDts.timeStampLow == 0
  - pesFilter.maxDts.timeStampHigh == 1
  - pesFilter.maxDts.timeStampLow == 0xFFFFFFFF
  - pesFilter.minPts.timeStampHigh == 0
  - pesFilter.minPts.timeStampLow == 0
  - pesFilter.maxPts.timeStampHigh == 1
  - pesFilter.maxPts.timeStampLow == 0xFFFFFFFF
- else *ppIPesFilter == Null

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.13.9 AllocatePesNotifier

Signature
```
uhErrorCode_t AllocatePesNotifier ( [out] uhITsDmxPesNotifier **ppIPesNotifier );
```

Qualifiers
- synchronous
- thread-safe

Description
Allocates a PES Notifier object and returns a pointer to the `uhITsDmxPesNotifier` interface of the allocated PES Notifier object.

Parameters

Table 837

Name	Description
`ppIPesNotifier`	Output parameter returning a pointer to the `uhITsDmxPesNotifier` interface of the allocated PES Notifier object.

Return Values
Only non-standard error values are listed:

Table 838

Name	Description
`UH_ERR_TSDMX_PES_FILTER_NOT_AVAILABLE`	The allocation failed due to a lack of available PES Notifier objects.

Precondition
- The `Stream` object - that is associated with the PID or TPH Filter object on which this interface operates - has (data type `uhTsDmx_Pes` and output type `ClientMemory`).
- Let `filter` be the `PidFilter` or `TphFilter` object on which this interface operates.
- `filter.nrPesFilters == 0`
- `filter.pIPesNotifier == Null`
- If the filter object on which this interface operates is a PID Filter
  - `filter.nrSectionFilters == 0`

Action
- `If demux.nrPesFilters < demux.maxNrPesFilters`
  - Modify `demux.nrPesFilters`
  - Modify `filter.pIPesNotifier`

Postcondition
- If `demux.nrPesFilters' < demux.maxNrPesFilters`
  - `*ppIPesNotifier` contains a pointer to the `uhI`TsDmxPesNotifier interface of the allocated PES Notifier object `pesNotifier`.
  - `demux.nrPesFilters == demux.nrPesFilters' + 1`
  - `filter.pIPesNotifier == *ppIPesNotifier`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    — pesNotifier.started == False
    — pesNotifier.filterMode == uhTsDmx_Continuous
— else *ppIPesNotifier == Null
```

8.1.13.5.14 uhITsDmxPidFilterNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a PID or a TPH Filter. The `Subscribe` function of the `uhITsDmxPidFilter` interface can be used to subscribe to these events.

Interface ID
`uuid( A279EA1C-971D-4af4-8AEA-B7B9D20DCBB1 )`

8.1.13.5.14.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                      cookie,
    [in] uhTsDmx_PidNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.13.5.14.2 OnStopped

Signature
```
[async] Void OnStopped ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_PidFilterStopped`. It is called in response to a `Stop` call to indicate that a filter object has stopped.

Parameters

Table 839

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
— Once received, the client will not receive any other event from the filter object on which the corresponding control interface operates before being started again, except for `uhTsDmx_PidFilterStopped` when an already stopped filter object is stopped again.

8.1.13.5.15 uhITsDmxDescramblerAssociation

Qualifiers
None.

Description
This interface of a PID or TPH Filter provides facilities to associate and disassociate a Descrambler to a PID or TPH Filter. It also provides a function to retrieve the scrambling status of a filtered component (e.g. a video PES).

Interface ID
`uuid( 52F74362-677A-4d02-BEEF-8CA0332B1024 )`

8.1.13.5.15.1 AssociateDescrambler

Signature
```
uhErrorCode_t AssociateDescrambler ( [in] uhITsDmxDescrambler *pIDescrambler );
```

Qualifiers
— synchronous

— thread-safe

Description
Associates an allocated `Descrambler` object with the PID or TPH Filter object on which this interface operates.

Parameters

Table 840

Name	Description
`pIDescrambler`	The pointer to the `uhITsDmxDescrambler` interface of the `Descrambler` object to associate.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
— `pIDescrambler` is a pointer to the `uhITsDmxDescrambler` interface of an allocated `Descrambler` object.
— Let `filter` be the `PidFilter` or `TphFilter` object on which this interface operates.
— `filter.pIDescrambler == Null`

Action
None.

Postcondition
`filter.pIDescrambler == pIDescrambler`

8.1.13.5.15.2 DisassociateDescrambler

Signature
```
uhErrorCode_t DisassociateDescrambler ( );
```

Qualifiers
— synchronous

— thread-safe

Description
Disassociates a `Descrambler` object from the PID or TPH Filter object on which this interface operates.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `filter` be the `PidFilter` or `TphFilter` object on which this interface operates.

```
filter.pIDescrambler == Null
```

Remarks

— The disassociated `Descrambler` object will not be deallocated until its reference count becomes 0.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.15.3 GetDescramblerAssociation

Signature
```
uhErrorCode_t GetDescramblerAssociation (
    [out] uhITsDmxDescrambler **ppIDescrambler );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether a `Descrambler` object is associated with the PID or TPH Filter object on which this interface operates.

Parameters

Table 841

Name	Description
`ppIDescrambler`	Output parameter returning a pointer to the `uhITsDmxDescrambler` interface of the associated Descrambler object. It is `NULL` when the PID or TPH Filter object on which this interface operates has no association with a `Descrambler` object.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `filter` be the `PidFilter` or `TphFilter` object on which this interface operates.
— `*ppIDescrambler == filter.pIDescrambler`

8.1.13.5.15.4 GetScramblingStatus

Signature
```
uhErrorCode_t GetScramblingStatus (
    [out] puhTsDmx_ScramblingStatus_t pScramblingStatus );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns whether the current stream - filtered by the PID or TPH Filter object on which this interface operates - is scrambled.

Parameters

Table 842

Name	Description
`pScramblingStatus`	Output parameter returning whether the current stream - filtered by the PID or TPH Filter object on which this interface operates - is scrambled.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `filter` be the `PidFilter` or `TphFilter` object on which this interface operates.
— `*pScramblingStatus == filter.`scramblingStatus

8.1.13.5.16 uhlTsDmxMatchMaskFilter

Qualifiers
None.

Description
This interface of TPH Filter allows match/mask control on filters. A mask is a bit pattern. A `1' at bit `x' of the mask means that the corresponding bit in the data to be filtered must match the value of bit `x' of the match value. A `0' at bit `x' of the mask means that the corresponding bit in the data to be filtered is a `don't care' with respect to whether it will be passed by the filter or not.

A transport packet passes the match/mask criteria if all bits of the packet match with their corresponding bit of the match value if they are not masked out.

Interface ID
`uuid( D5B75802-FB6C-43E4-BAD2-A4292A0F450B )`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.16.1 GetMatchMask

Signature

```
uhErrorCode_t GetMatchMask (
    [in] UInt32                 size,
    [out, size_is(size)] pUInt8 pMatchValue,
    [out, size_is(size)] pUInt8 pMaskValue );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the current match and mask value of the TPH Filter.

Parameters

Table 843

Name	Description
`size`	The size - in bytes - of `*pMatchValue` and `*pMaskValue`.
`pMatchValue`	Output parameters returning the current match value.
`pMaskValue`	Output parameters returning the current mask value.

Return Values

Standard.

Precondition

— `size == ceiling(demux.tphFilterDepth/8)`

Action

None.

Postcondition

— For each `i` in `{0 .. size-1}`
  — `pMatchValue [i] == tphFilter.pMatchValue[i]`
  — `pMaskValue [i] == tphFilter.pMaskValue[i]`

8.1.13.5.16.2 SetMatchMask

Signature

```
uhErrorCode_t SetMatchMask (
    [in] UInt32                       size,
    [in, size_is(size)] const pUInt8 pMatchValue,
    [in, size_is(size)] const pUInt8 pMaskValue );
```

Qualifiers

— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— single-threaded

Description
Sets the match value and mask value of the TPH Filter.

Parameters

Table 844

Name	Description
`size`	The size - in bytes - of `*pMatchValue` and `*pMaskValue`.
`pMatchValue`	Pointer to the match value to be set.
`pMaskValue`	Pointer to the mask value to be set.

Return Values
Standard.

Precondition
— Let `tphFilter` be the `TphFilter` object on which this interface operates.
— `1 ≤ size ≤ ceiling(demux.tphFilterDepth/8)`

Action
— Modify `tphFilter.`pMatchValue
— Modify `tphFilter.`pMaskValue

Postcondition
— For each `i` in `{0 .. size-1}`
  — `tphFilter.pMatchValue[i] == pMatchValue [i]`
  — `tphFilter.pMaskValue[i] == pMaskValue [i]`

Remarks
— Figure 89 show the bit-mask for a TPH Filter. If a bit is excluded, it means that no match and mask filter criteria can be set on this bit.

— The sync_byte, the PID, and the transport_error_indicator bit are excluded from the mask and match field. See Figure 89.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

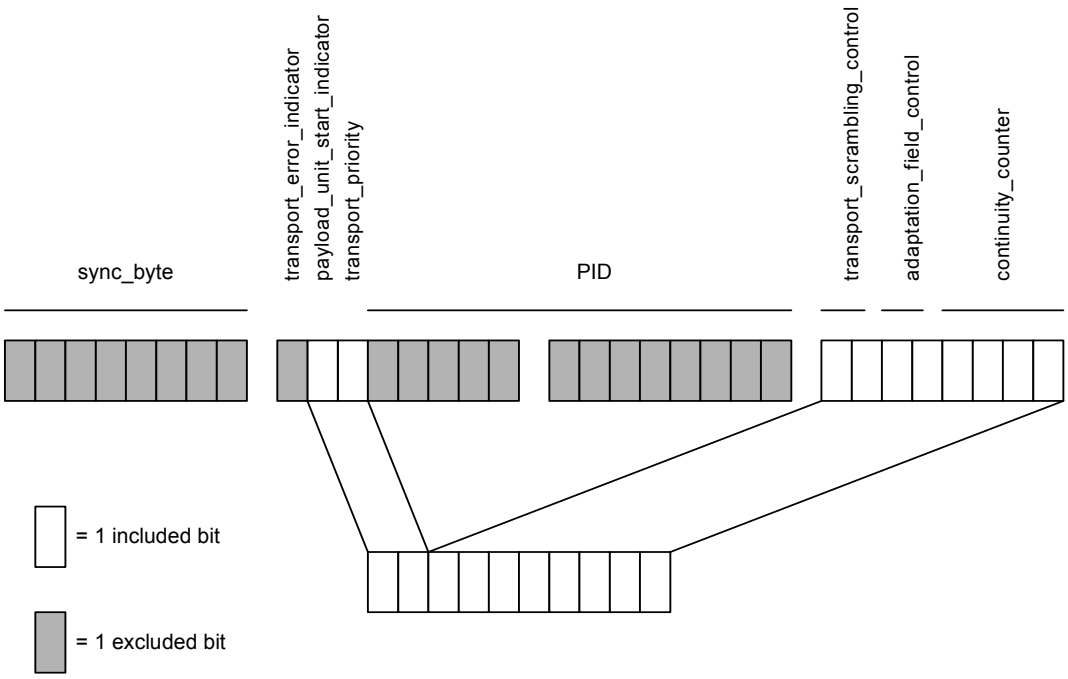


Figure 89 — The 10-bit mask for TPH Filters

8.1.13.5.16.3 GetFilterDepth

Signature
```
uhErrorCode_t GetFilterDepth ( [out] pUInt32 pFilterDepth );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the filter depth - in **bits** - of the TPH Filter object.

Parameters

Table 845

Name	Description
`pFilterDepth`	Output parameter returning the filter depth - in **bits** - of a TPH Filter object.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `*pFilterDepth == demux.tphFilterDepth`

8.1.13.5.17 uhITsDmxSectionFilter

Qualifiers
None.

Description
This interface of a Section Filter provides facilities to start and to stop the filter. It provides also facilities to retrieve and to set the filter mode and allows match/mask control on the filter. It also provides a `Subscribe` and `Unsubscribe` function on notifications of events that occur in the Section Filter.

This interface also provides facilities to check the integrity of data. It allows CRC checks, setting and retrieval of the CRC mode.

Interface ID
`uuid( B44FD7D1-D24B-4C50-9609-1DE7EA08CD23 )`

8.1.13.5.17.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhITsDmxSectionFilterNtf        *pINotify,
    [in] UInt32                          cookie,
    [in] uhTsDmx_SectionNotifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.13.5.17.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhITsDmxSectionFilterNtf        *pINotify,
    [in] UInt32                          cookie,
    [in] uhTsDmx_SectionNotifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.13.5.17.3 Start

Signature
```
uhErrorCode_t Start ( );
```

Qualifiers
— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Starts the Section Filter object on which this interface operates.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Action
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.
— Modify `sectionFilter.started`

Postcondition
— `sectionFilter.started == True`

8.1.13.5.17.4 Stop

Signature
```
uhErrorCode_t Stop ( );
```

Qualifiers
— asynchronous

— single-threaded

Description
Sends a stop request to the Section Filter object on which this interface operates.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Asynchronous **Action**
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.
— Modify `sectionFilter.started`
— Notify `OnStopped(*)`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Asynchronous Postcondition
— `sectionFilter.started == False`

8.1.13.5.17.5 GetFilterDepth

Signature
```
uhErrorCode_t GetFilterDepth ( [out] pUInt32 pFilterDepth );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the filter depth - in bytes - of Section Filters.

Parameters

Table 846

Name	Description
`pFilterDepth`	Output parameter returning the filter depth - in bytes - of Section Filters.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.
— `*pFilterDepth == demux.sectionFilterDepth`

8.1.13.5.17.6 GetFilterMode

Signature
```
uhErrorCode_t GetFilterMode ( [out] puhTsDmx_FilterMode_t pFilterMode );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current filter mode of the Section Filter object.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 847

Name	Description
`pFilterMode`	Output parameter returning the current filter mode.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.
— `*pFilterMode == sectionFilter.`filterMode

8.1.13.5.17.7 SetFilterMode

Signature
```
uhErrorCode_t SetFilterMode ( [in] uhTsDmx_FilterMode_t filterMode );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the filter mode of the Section Filter object.

Parameters

Table 848

Name	Description
`filterMode`	The filter mode.

Return Values
Standard.

Precondition
`True`

Action
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.
— Modify `sectionFilter.filterMode`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`sectionFilter.filterMode == filterMode`

8.1.13.5.17.8 GetMatchMask

Signature
```
uhErrorCode_t GetMatchMask (
    [in] UInt32                 size,
    [out, size_is(size)] pUInt8 pModeValue,
    [out, size_is(size)] pUInt8 pMatchValue,
    [out, size_is(size)] pUInt8 pMaskValue );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current mode value, match value, and mask value of the Section Filter.

Parameters

Table 849

Name	Description
`size`	The size - in bytes - of `*pModeValue`, `*pMatchValue` and `*pMaskValue`.
`pModeValue`	Output parameters returning the start address of the current mode value.
`pMatchValue`	Output parameters returning the start address of the current match value.
`pMaskValue`	Output parameters returning the start address of the current mask value.

Return Values
Standard.

Precondition
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.
— `size == demux.sectionFilterDepth`

Action
None.

Postcondition
— For each `i` in `{0 .. size -1}`
  — `pModeValue[i] == sectionFilter.pModeValue[i]`
  — `pMatchValue[i] == sectionFilter.pMatchValue[i]`
  — `pMaskValue[i] == sectionFilter.pMaskValue[i]`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.17.9 SetMatchMask

Signature

```
uhErrorCode_t SetMatchMask (
    [in] UInt32                       size,
    [in, size_is(size)] const pUInt8 pModeValue,
    [in, size_is(size)] const pUInt8 pMatchValue,
    [in, size_is(size)] const pUInt8 pMaskValue );
```

Qualifiers

— synchronous

— single-threaded

Description

Sets the mode value, match value, and mask value of the Section Filter.

Parameters

Table 850

Name	Description
`size`	The size - in bytes - of `*pModeValue`, `*pMatchValue` and `*pMaskValue`.
`pModeValue`	Pointer to the mode value.
`pMatchValue`	Pointer to the match value.
`pMaskValue`	Pointer to the mask value.

Return Values

Standard.

Precondition

— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.

— `1 ≤ size ≤ demux.sectionFilterDepth`

Action

— Modify `sectionFilter.pModeValue`

— Modify `sectionFilter.pMatchValue`

— Modify `sectionFilter.pMaskValue`

Postcondition

— For each `i` in `{0 .. size-1}`

  — `sectionFilter.pModeValue[i] == pModeValue [i]`

  — `sectionFilter.pMatchValue[i] == pMatchValue [i]`

  — `sectionFilter.pMaskValue[i] == pMaskValue [i]`

Remarks

— Figure 90 shows the byte-mask for the Section Filter. If a bit is excluded, it means that no match and mask filter criteria can be set on this bit.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

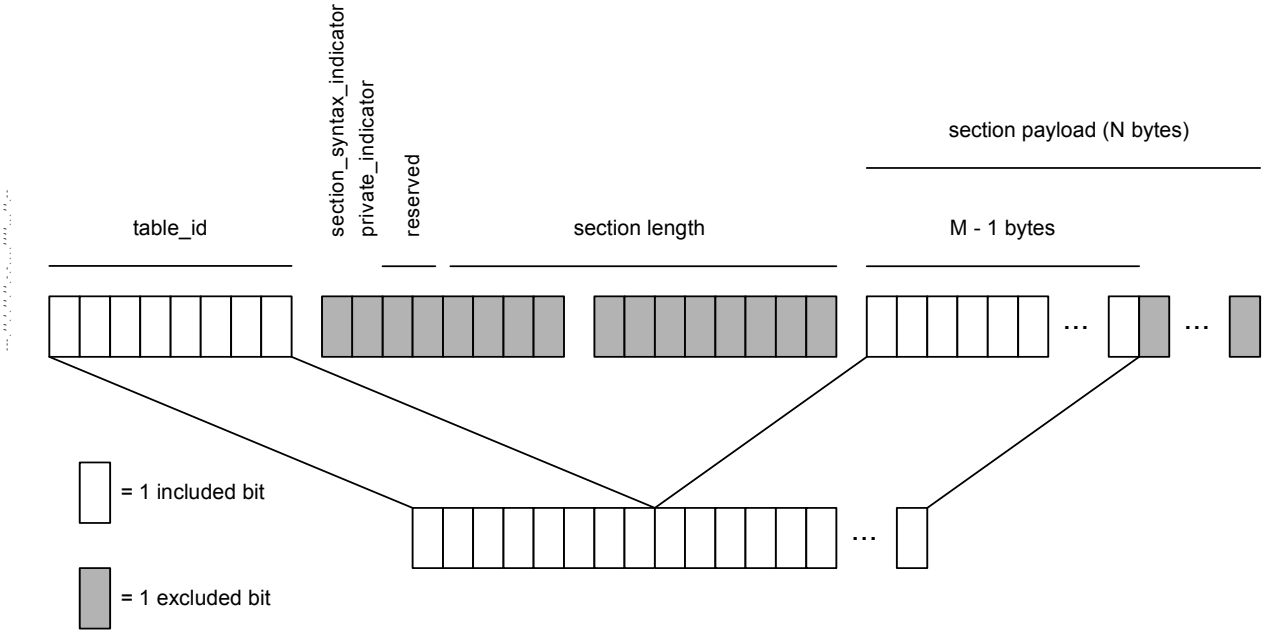


Figure 90 — M byte-mask for Section Filters (M = the depth of Section Filters)

— A `1' at bit `x' of the mode value means that negative filtering should be applied to the corresponding bit in the data. A `0' at bit `x' of the mode value means that positive filtering should be applied to the corresponding bit in the data. Note that positive and negative filtering can be selected per bit. So, on the data entity on which the match, mask, and mode values apply, positive and negative filtering can be mixed.

8.1.13.5.17.10 GetCrcMode

Signature

```
uhErrorCode_t GetCrcMode ( [out] puhTsDmx_CrcMode_t pCrcMode );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the current CRC mode of the Section Filter object.

Parameters

Table 851

Name	Description
`pCrcMode`	Output parameter returning the current CRC mode.

Return Values

Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.
— `*pCrcMode == sectionFilter.`crcMode

8.1.13.5.17.11 SetCrcMode

Signature
```
uhErrorCode_t SetCrcMode ( [in] uhTsDmx_CrcMode_t crcMode );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the CRC mode of the Section Filter object.

Parameters

Table 852

Name	Description
`crcMode`	The CRC mode.

Return Values
Standard.

Precondition
`True`

Action
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.
— Modify `sectionFilter.crcMode`

Postcondition
— `sectionFilter.`crcMode `==` crcMode

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.17.12 CheckCrc

Signature

```
uhErrorCode_t CheckCrc (
    [in] UInt32                                    primarySectionSize,
    [in, size_is(primarySectionSize)] const pUInt8 pPrimarySectionStartAddress,
    [in] UInt32                                    secondarySectionSize,
    [in, size_is(secondarySectionSize)]
                                const pUInt8 pSecondarySectionStartAddress,
    [out] puhTsDmx_CrcStatus_t                     pCrc );
```

Qualifiers

— synchronous

— thread-safe

Description

Checks the CRC of the provided section. The check of the CRC can be done on some individual sections e.g. in case CRC checking on all sections is too expensive.

Parameters

Table 853

Name	Description
`primarySectionSize`	The size in bytes of valid section data excluding the second part of the section in case a wrap-around (in the buffer containing this data) occurred.
`pPrimarySectionStartAddress`	Pointer to a 4 byte aligned buffer with `primarySectionSize` bytes of valid section data.
`secondarySectionSize`	The size in bytes of the second part of valid section data in case a wrap-around (in the buffer containing this data) occurred.
`pSecondarySectionStartAddress`	Pointer to a 4 byte aligned buffer with `secondarySectionSize` bytes of the second part of valid section data in case a wrap-around (in the buffer containing this data) occurred.
`pCrc`	Output parameter returning the result of the CRC check to be returned.

Return Values

Standard.

Precondition

— `secondarySectionSize` is 0 for linear buffers.

— `secondarySectionSize` is 0 for circular buffers in case no wrap-around has occurred.

— If wrap-around occurred in the buffer in this section, `pPrimarySectionStartAddress` and `pSecondarySectionStartAddress` point to a previously filtered, notified, and not yet freed section. If no wrap-around occurred, only `pPrimarySectionStartAddress` points to a previously filtered, notified, and not yet freed section and `pSecondarySectionStartAddress` is irrelevant.

Action

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— Let `sectionFilter` be the `SectionFilter` object on which this interface operates.

— `*pCrc == sectionFilter.crc`

8.1.13.5.18 uhITsDmxSectionFilterNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a Section Filter. The `Subscribe` function of the `uhITsDmxSectionFilter` can be used to subscribe to these events.

Interface ID
`uuid( 2B86BF80-7D24-4906-8359-EAE9DAF624EB )`

8.1.13.5.18.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                          cookie,
    [in] uhTsDmx_SectionNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.13.5.18.2 OnStopped

Signature
```
[async] Void OnStopped ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_SectionFilterStopped`. It is called in response to a `Stop` call to indicate that a Section Filter object has stopped.

Parameters

Table 854

Name	Description
`cookie`	The value that was passed by the client at subscription time.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
Once received, the client will not receive any other event from the Section Filter object on which the corresponding control interface operates before being started again, except for `uhTsDmx_SectionFilterStopped` when an already stopped filter object is stopped again. This event is fired after a section has been filtered during a `one shot' operation. In this case, the Section Filter object is stopped on reception of a complete data unit.

8.1.13.5.18.3 OnSectionArrived

Signature
```
[async] Void OnSectionArrived (
    [in] UInt32 cookie,
    [in] UInt32 dataCookie,
    [in] UInt32 primarySectionSize,
    [in, size_is(primarySectionSize)] const pUInt8 pPrimarySectionStartAddress,
    [in] UInt32 secondarySectionSize,
    [in,          size_is(secondarySectionSize)]          const          pUInt8
pSecondarySectionStartAddress,
    [in] uhTsDmx_CrcStatus_t crc);
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_SectionArrived`. It is called when a complete MPEG section has been filtered and written to a buffer. Complete means starting from and including the `table_id' to the last byte of this section including the CRC field, i.e. `section_length' + 3 bytes of this section, if present.

Parameters

Table 855

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`dataCookie`	Value that should be returned as is (without any translation) when releasing a lock on this notified section data (via the `Free` function of the `uhITsDmxStreamBuffer` interface).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
primarySectionSize	The size in bytes of valid section data excluding the second part of the section in case a wrap-around (in the buffer containing this data) occured.
pPrimarySectionStartAddress	Pointer to a buffer with primarySectionSize bytes of valid section data.
secondarySectionSize	The size in bytes of the second part of valid section data in case a wrap-around (in the buffer containing this data) occured. It is 0 for linear buffers. It is 0 for circular buffers in case no wrap-around has occurred.
pSecondarySectionStartAddress	Pointer to a buffer with secondarySectionSize bytes of the second part of valid section data in case a wrap-around (in the buffer containing this data) occurred.
crc	The result of the CRC check on this notified section data.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

Remarks
The data notified by this function is only freed when all notified clients have released the lock on this data by calling the Free function on the uhITsDmxStreamBuffer interface of the ClientStream object with which the Section Filter object on which the corresponding control interface operates is (indirectly) associated.

8.1.13.5.19 uhITsDmxPesNotifier

Qualifiers
None.

Description
This interface of a PES Notifier or a PES Filter provides facilities to start and to stop the object on which this interface operates. It provides also facilities to retrieve and to set the filter mode. It also provides a Subscribe and Unsubscribe function on notifications of events that occur in the notifier.

Interface ID
uuid(5D2C7794-C6BA-44be-9003-BF4EC697548A)

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.19.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhITsDmxPesNotifierNtf      *pINotify,
    [in] UInt32                      cookie,
    [in] uhTsDmx_PesNotifications_t notifs );
```

Qualifiers

— subscribe-function

8.1.13.5.19.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhITsDmxPesNotifierNtf      *pINotify,
    [in] UInt32                      cookie,
    [in] uhTsDmx_PesNotifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.13.5.19.3 Start

Signature

```
uhErrorCode_t Start ( );
```

Qualifiers

— synchronous

— single-threaded

Description

Starts the PES Notifier or PES Filter object on which this interface operates.

Parameters

None.

Return Values

Standard.

Precondition

`True`

Action

— Let `pesNotifierFilter` be the `PesNotifier` or `PesFilter` object on which this interface operates.

— Modify `pesNotifierFilter.started`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `pesNotifierFilter.started == True`

8.1.13.5.19.4 Stop

Signature
`uhErrorCode_t Stop ( );`

Qualifiers
— asynchronous

— single-threaded

Description
Sends a stop request to the PES Notifier or PES Filter object on which this interface operates.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Asynchronous Action
— Let `pesNotifierFilter` be the `PesNotifier` or `PesFilter` object on which this interface operates.
— Modify `pesNotifierFilter.started`
— Notify `OnStopped(*)`

Asynchronous Postcondition
— `pesNotifierFilter.started == False`

8.1.13.5.19.5 GetFilterMode

Signature
`uhErrorCode_t GetFilterMode ( [out] puhTsDmx_FilterMode_t pFilterMode );`

Qualifiers
— synchronous

— thread-safe

Description
Returns the current filter mode of the PES Notifier or PES Filter object on which this interface operates.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 856

Name	Description
pFilterMode	Output parameter returning the current filter mode.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— Let pesNotifierFilter be the PesNotifier or PesFilter object on which this interface operates.

— *pFilterMode == pesNotifierFilter.filterMode

8.1.13.5.19.6 SetFilterMode

Signature
```
uhErrorCode_t SetFilterMode ( [in] uhTsDmx_FilterMode_t filterMode );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the filter mode of the PES Notifier or PES Filter object on which this interface operates.

Parameters

Table 857

Name	Description
filterMode	The filter mode.

Return Values
Standard.

Precondition
— If the object on which this interface operates is a PES Filter, the output type of the associated Stream object (via the associated PID or TPH Filter object) is ClientMemory.

Action
— Let pesNotifierFilter be the PesNotifier or PesFilter object on which this interface operates.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Modify `pesNotifierFilter.filterMode`

Postcondition
— `pesNotifierFilter.filterMode == filterMode`

8.1.13.5.20 uhITsDmxPesNotifierNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a PES Filter or a PES Notifier; the `Subscribe` function of the `uhITsDmxPesNotifier` interface can be used to subscribe to these events.

Interface ID
`uuid( 81E5B522-4A8B-444F-90B8-7AE1282059B1 )`

8.1.13.5.20.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                    cookie,
    [in] uhTsDmx_PesNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.13.5.20.2 OnStopped

Signature
```
[async] Void OnStopped ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_PesNotifierStopped`. It is called in response to a `Stop` call to indicate that a PES Filter or a PES Notifier object has stopped.

Parameters

Table 858

Name	Description
`cookie`	The value that was passed by the client at subscription time.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
Once received, the client will not receive any other event from the filter object on which the corresponding control interface operates before being started again, except for `uhTsDmx_PesNotifierStopped` when an already stopped filter object is stopped again. This event is also fired after a PES packet has been filtered during a `one shot' operation. In this case, the filter object on which this interface operates is stopped on reception of a complete data unit.

8.1.13.5.20.3 OnCompletePacketArrived

Signature
```
[async] Void OnCompletePacketArrived (
    [in] UInt32 cookie,
    [in] UInt32 dataCookie,
    [in] UInt32 primaryPesSize,
    [in, size_is(primaryPesSize)] const pUInt8 pPrimaryPesStartAddress,
    [in] UInt32 secondaryPesSize,
    [in, size_is(secondaryPesSize)] const pUInt8 pSecondaryPesStartAddress );
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_PesCompletePacketArrived`. It is called when a complete PES packet has been filtered and written to a buffer. Complete means starting from and including the PES packet header to the last byte of this PES packet.

Parameters

Table 859

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`dataCookie`	Value that is returned as is (without any translation) when releasing a lock on this notified complete PES packet (via the `Free` function of the `uhITsDmxStreamBuffer` interface).
`primaryPesSize`	The size in bytes of valid PES packet data excluding the second part of the PES packet in case a wrap-around (in the buffer containing this data) occured.
`pPrimaryPesStartAddress`	Pointer to a buffer with `primaryPesSize` bytes of valid PES packet data.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
secondaryPesSize	The size in bytes of the second part of valid PES packet data in case a wrap-around (in the buffer containing this data) occured. It is 0 for linear buffers. It is 0 for circular buffers in case no wrap-around has occurred.
pSecondaryPesStartAddress	Pointer to a buffer with secondaryPesSize bytes of the second part of valid PES packet data in case a wrap-around (in the buffer containing this data) occurred.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

Remarks
The data notified by this function is only freed when all notified clients have released the lock on this data by calling the Free function on the uhITsDmxStreamBuffer interface of the ClientStream object with which the PES Filter or PES Notifier object on which the corresponding control interface operates is (indirectly) associated.

8.1.13.5.20.4 OnPartialPacketArrived

Signature
```
[async] Void OnPartialPacketArrived (
    [in] UInt32 cookie,
    [in] UInt32 dataCookie,
    [in] UInt32 primaryPesSize,
    [in, size_is(primaryPesSize)] const pUInt8 pPrimaryPesStartAddress,
    [in] UInt32 secondaryPesSize,
    [in, size_is(secondaryPesSize)] const pUInt8 pSecondaryPesStartAddress );
```

Qualifiers
None.

Description
This function is only called when subscribed to uhTsDmx_PesPartialPacketArrived. It is called when a part of a PES packet has been filtered and written to a buffer. The size of this block equals the actual granularity.

Parameters

Table 860

Name	Description
cookie	The value that was passed by the client at subscription time.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
dataCookie	Value that is returned as is (without any translation) when releasing a lock on this notified partial PES packet (via the Free function of the uhITsDmxStreamBuffer interface).
primaryPesSize	The size in bytes of valid partial PES packet data excluding the second part of the partial PES packet in case a wrap-around (in the buffer containing this data) occured.
pPrimaryPesStartAddress	Pointer to a buffer with primaryPesSize bytes of valid partial PES packet data.
secondaryPesSize	The size in bytes of the second part of valid partial PES packet data in case a wrap-around (in the buffer containing this data) occured. It is 0 for linear buffers. It is 0 for circular buffers in case no wrap-around has occurred.
pSecondaryPesStartAddress	Pointer to a buffer with secondaryPesSize bytes of the second part of valid partial PES packet data in case a wrap-around (in the buffer containing this data) occurred.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

Remarks
— The data notified by this function is only freed when all notified clients have released the lock on this data by calling the Free function on the uhITsDmxStreamBuffer interface of the ClientStream object with which the PES Filter or PES Notifier object on which the corresponding control interface operates is (indirectly) associated.

8.1.13.5.20.5 OnPacketCorrupt

Signature
```
[async] Void OnPacketCorrupt ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is only called when subscribed to uhTsDmx_PesPacketCorrupt. It is called when a partial or a complete PES packet has been flagged as corrupt due to a corruption of at least one transport packet that contains a part of this PES packet.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 861

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.13.5.20.6 OnStreamIdChanged

Signature

```
[async] Void OnStreamIdChanged (
    [in] UInt32 cookie,
    [in] UInt8  streamId );
```

Qualifiers
None.

Description
This function is only called when subscribed to `uhTsDmx_PesStreamIdChanged`. It is called when the `stream_id' has changed. If the PES Filter or the PES Notifier starts, this event will be fired as well on the arrival of the first PES packet. This can be viewed as a transition from an undefined to a defined `stream_id'.

Parameters

Table 862

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`streamId`	The new `stream_id' value.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

8.1.13.5.21 uhITsDmxPesFilter

Qualifiers
None.

Description
This interface of a PES Filter provides facilities for match/mask control on the filter. This interface also provides facilities to retrieve and set DTS/PTS ranges that are used as a filter criterion.

Interface ID
`uuid( FFD29485-242D-459B-A917-46BD8B00CD62 )`

8.1.13.5.21.1 GetFilterDepth

Signature
```
uhErrorCode_t GetFilterDepth ( [out] pUInt32 pFilterDepth );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns the filter depth - in bytes - of PES Filters.

Parameters

Table 863

Name	Description
`pFilterDepth`	Output parameter returning the filter depth - in bytes - of PES Filters.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
- Let `pesFilter` be the `PesFilter` object on which this interface operates.
- `*pFilterDepth == demux.pesFilterDepth`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.21.2 GetMatchMask

Signature

```
uhErrorCode_t GetMatchMask (
    [in] UInt32                 size,
    [out, size_is(size)] pUInt8 pMatchValue,
    [out, size_is(size)] pUInt8 pMaskValue );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the current match and mask value of the PES Filter.

Parameters

Table 864

Name	Description
`size`	The size - in bytes - of `*pMatchValue` and `*pMaskValue`.
`pMatchValue`	Output parameters returning the current match value.
`pMaskValue`	Output parameters returning the current mask value.

Return Values

Standard.

Precondition

— Let `pesFilter` be the `PesFilter` object on which this interface operates.
— `size == demux.pesFilterDepth`

Action

None.

Postcondition

— For each `i` in `{0 .. size-1}`
  — `pMatchValue[i] == pesFilter.pMatchValue[i]`
  — `pMaskValue[i] == pesFilter.pMaskValue[i]`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.21.3 SetMatchMask

Signature

```
uhErrorCode_t SetMatchMask (
    [in] UInt32                       size,
    [in, size_is(size)] const pUInt8 pMatchValue,
    [in, size_is(size)] const pUInt8 pMaskValue );
```

Qualifiers

— synchronous

— single-threaded

Description

Sets the match value and mask value of the PES Filter.

Parameters

Table 865

Name	Description
`size`	The size - in bytes - of `*pMatchValue` and `*pMaskValue`.
`pMatchValue`	Pointer to the requested match value.
`pMaskValue`	Pointer to the requested mask value.

Return Values

Standard.

Precondition

— Let `pesFilter` be the `PesFilter` object on which this interface operates.

— `1 ≤ size ≤ demux.pesFilterDepth`

Action

— Modify `pesFilter.pMatchValue`

— Modify `pesFilter.pMaskValue`

Postcondition

— For each `i` in `{0 .. size-1}`

```
pesFilter.pMatchValue[i] == pMatchValue[i]

pesFilter.pMaskValue[i] == pMaskValue[i]
```

Remarks

— Figure 91 shows the byte-mask for the PES Filter. If a bit is excluded, it means that no match and mask filter criteria can be set on this bit.

— Filtering on the trick mode field can be applied in case the match/mask bit on the `DSM_trick_mode_flag` is set in such a way that the presence of this trick mode field can be determined for each incoming PES packet. See Figure 91.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

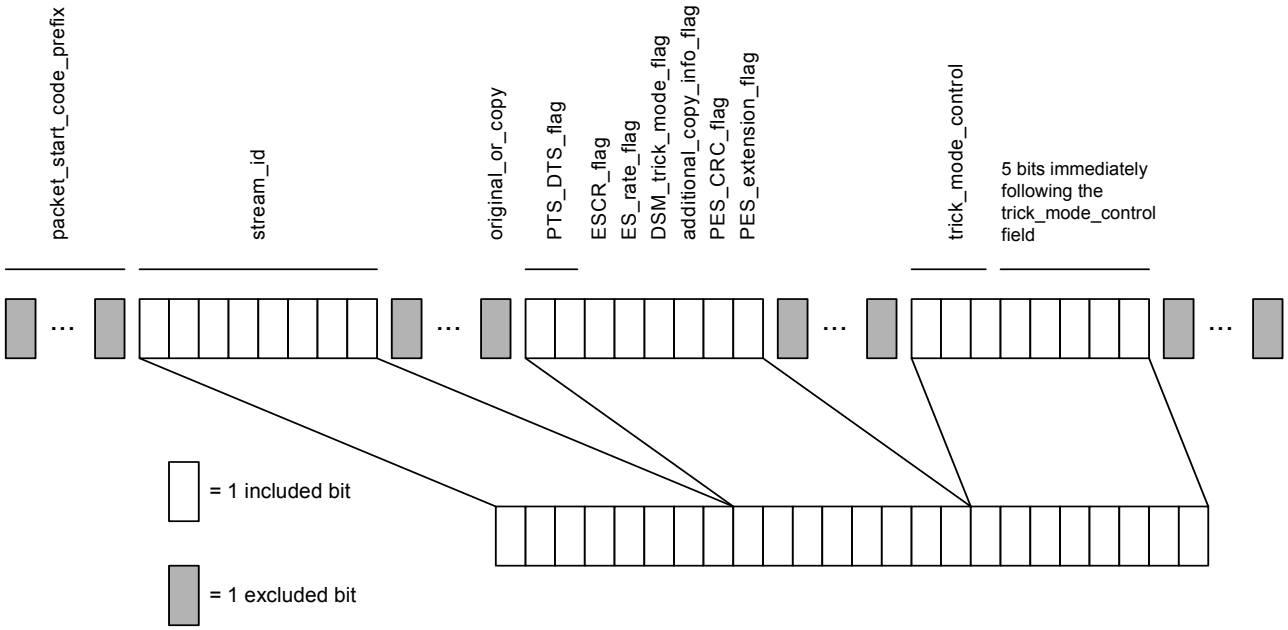


Figure 91 — bit-mask for PES Filters

8.1.13.5.21.4 GetDtsRange

Signature

```
uhErrorCode_t GetDtsRange(
    [out] puhTimeStamp_t pMinDts,
    [out] puhTimeStamp_t pMaxDts );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the minimum and maximum DTS value of the current range for DTS filtering of the PES Filter object.

Parameters

Table 866

Name	Description
`pMinDts`	Output parameter returning the minimum DTS value.
`pMaxDts`	Output parameter returning the maximum DTS value.

Return Values

Standard.

Precondition

`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— Let `pesFilter` be the PesFilter object on which this interface operates.
— `*pMinDts == pesFilter.minDts`
— `*pMaxDts == pesFilter.maxDts`

8.1.13.5.21.5 SetDtsRange

Signature
```
uhErrorCode_t SetDtsRange (
    [in] uhTimeStamp_t minDts,
    [in] uhTimeStamp_t maxDts );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the minimum and maximum DTS value of the range for DTS filtering of the PES Filter object: PES packets that contain a DTS value will only pass the filter if the DTS values are within the specified range.

Parameters

Table 867

Name	Description
`minDts`	The minimum DTS value.
`maxDts`	The maximum DTS value.

Return Values
Standard.

Precondition
`True`

Action
— Let `pesFilter` be the `PesFilter` object on which this interface operates.
— Modify `pesFilter.minDts`
— Modify `pesFilter.maxDts`

Postcondition
— `pesFilter.minDts == minDts`
— `pesFilter.maxDts == maxDts`

Remarks
— PES packets that do not contain a DTS will pass the filter criteria for DTS filtering.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.13.5.21.6 GetPtsRange

Signature

```
uhErrorCode_t GetPtsRange (
    [out] puhTimeStamp_t pMinPts,
    [out] puhTimeStamp_t pMaxPts );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the minimum and maximum PTS value of the current range for PTS filtering of the PES Filter object.

Parameters

Table 868

Name	Description
`pMinPts`	Output parameter returning the minimum PTS value.
`pMaxPts`	Output parameter returning the maximum PTS value.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— Let `pesFilter` be the `PesFilter` object on which this interface operates.

— `*pMinPts == pesFilter.minPts`

— `*pMaxPts == pesFilter.maxPts`

8.1.13.5.21.7 SetPtsRange

Signature

```
uhErrorCode_t SetPtsRange (
    [in] uhTimeStamp_t minPts,
    [in] uhTimeStamp_t maxPts );
```

Qualifiers

— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Sets the minimum and maximum PTS value of the range for PTS filtering of the PES Filter object: PES packets that contain a PTS value will only pass the filter if the PTS values are within the specified range.

Parameters

Table 869

Name	Description
`minPts`	The minimum PTS value.
`maxPts`	The maximum PTS value.

Return Values
Standard.

Precondition
— `True`

Action
— Let `pesFilter` be the `PesFilter` object on which this interface operates.
— Modify `pesFilter.minPts`
— Modify `pesFilter.maxPts`

Postcondition
— `pesFilter.minPts == minPts`
— `pesFilter.maxPts == maxPts`

Remarks
— PES packets that do not contain a PTS will pass the filter criteria for PTS filtering.

8.1.14 UhlTsMux

8.1.14.1 Concepts

8.1.14.1.1 Introduction

This document describes the API of a multiplexer for an MPEG-2 transport stream.

The function of a TS Mux is to input a transport stream, allow the client to control the rate and number of insertions per section into this transport stream and output this multiplexed transport stream.

8.1.14.1.2 Multiplexing

The TS Mux is responsible for inserting the PSI section data from the client into the TS (with out time stamps) or TS_{ts} (with time stamps) input by an external component (e.g. TS Demux output). The TS/TS_{ts} input normally contains only the PES data from a specific program (e.g. SPTS) and no PSI section data corresponding to that program. The client, who receives the PSI section data of the multi-program TS through the demultiplexing process (e.g. through TS Demux notifications), modifies the original PSI data to make it match the characteristics of the TS/TS_{ts} being input to the TS Mux. The TS Mux role is to multiplex this modified PSI data, provided as sections by the client, into the input TS/TS_{ts}. The TS Mux first converts the sections into transport packets of the appropriate PIDs (also provided by the client), calculates time-stamps to be appended

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

to these transport packets if the input is TS_{ts} and then insert these into the input stream. The TS Mux then streams out this section data multiplexed $TS/TS_{ts.}$

Next to PSI section data, the client can also insert other section data.

— An 'insertion point' is defined as a point suitable for a transport packet insertion if the time-stamp values associated with the two transport packets on either side of this point need not be changed to accommodate the transport packet to be inserted. Basically, this also means that the difference between time-stamp values, defined here as the 'time-stamp distance', between the inserted packet and the two transport packets on either side of it should satisfy the 'minimum time-stamp distance' requirement.

— For example let's assume the time-stamp values are derived from a 27 MHz clock counter and each transport packet in the output stream from the TS Mux is being consumed and/or transmitted at an 'output bit-rate'[21]. Let this output bit-rate be 24 Mbps. Also, each packet is consumed and/or transmitted only if the associated time-stamp matches the value of a clock counter running at 27 MHz. Then the 'minimum time-stamp distance' between the two consecutive transport packets should be greater than the

— ((transport-packet-size-in-bits) x (output bit-rate)) / (1-tick-time-of-the-clock-counter).

— With the above example values for clock counter frequency and output bit-rate speed:

— Minimum time-stamp distance = ((188 x 8) x $(1/24*10^6)$) / $(1/27*10^6)$ = 1692.

— In the above calculation 188 is the number of bytes in a MPEG-2 transport packet.

— In case, the time-stamps are derived from a 13.5 MHz clock:

— Minimum time-stamp distance = ((188 x 8) x $(1/24*10^6)$) / $(1/13.5*10^6)$ = 846.

— The 'minimum time-stamp distance' value used by the TS Mux is dependent on the hardware platform configuration in which it is used and cannot be controlled by the client.

— The input stream has time-stamps associated with each transport packet. Using the time-stamp information the TS Mux detects a suitable insertion point in the stream into which the transport packet containing a section can be inserted. For repeated insertion of a section the same technique is followed to find suitable insertion points in the stream wherein two such insertion points are separated by the cycle-period (insertion time interval) specified for this section by the client. The cycle-period in this case is actually the difference between the time-stamp values (translated into the cycle-period units) of the two consecutive insertion points.

— If a section spans more than one transport packet then the first packet is inserted at the first suitable insertion point found and the remaining are inserted at the suitable insertion points immediately following this first point. This also means that the cycle-period specified for repeated section insertion is only maintained between two consecutive transport packets containing the beginning of the section data.

— The time-stamp value attached to the transport packet containing a section is equal to the time-stamp value calculated for the insertion point at which the transport packet is being inserted. This time-stamp value is always calculated by taking the minimum time-stamp distance requirement into consideration. In addition, the time-stamp values of the original transport packets in the input stream are never altered.

— The cycle-period between repeated section insertion is not guaranteed to be always equal to the client set value. This is because the insertion is done on a best-effort basis in order to maintain the minimum time-stamp distance requirement between any two transport packets in the multiplexed transport stream output.

---

21 For the transmission scenario, we can consider a specialized DMA device as an example. This device will read a transport packet output by the TS Mux, match the time-stamp associated with the packet with it's clock counter and transmit it out as a bit-stream, at the 'output bit-rate', if the match result is positive.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— A section, spanning over two transport packets, being inserted two times at suitable insertion points in a TS_{ts} is illustrated in Figure 92 below.

The Figure 93 shows the actual meaning of cycle-period between two insertions of the same section. The output transport stream from the TS Mux is being transmitted and/or consumed at a rate determined by each packet's time-stamp value. That is, each packet output by TS Mux is transmitted and/or consumed only if its time-stamp value matches the clock counter (which has the same frequency as the one used for generating the time-stamps) value. Therefore, in the transmission and/or consumption time-domain, the time gap between two transport packets containing the same section will then be equal to the cycle-period.

Section insertion into TS

For the timed and repeated insertion of a section into the TS the following conceptual points hold good:

— The input stream has transport packets without time-stamps associated with each packet. The TS Mux will insert transport packets containing sections at appropriate insertion points on a best-effort basis. The algorithm used for the detection of appropriate insertion points is an implementation defined behavior. For repeated insertion of a section the next insertion point used will be based on the cycle-period required between the next insertion point and the previous one.

— If a section spans more than one transport packet then the first packet is inserted at the first available insertion point in the stream and the remaining packets are inserted immediately following this first point. This also means that the cycle-period specified for repeated section insertion is only maintained between two consecutive transport packets containing the beginning of section data.

— The cycle-period between repeated section insertions is not guaranteed to be always equal to the client set value and is dependent on the implementation defined behavior.

— If the TS output by the TS Mux is transmitted and/or consumed at the correct bit-rate then the cycle-period between the same two sections extracted from this TS will be correct. Actually, if the transmission and/or consumption rate is not correct then depending on the transmission and/or consumption rate an extracted section may arrive sooner or later than the previously extracted section.

A section, spanning over two transport packets, being inserted into TS is illustrated in Figure 92.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

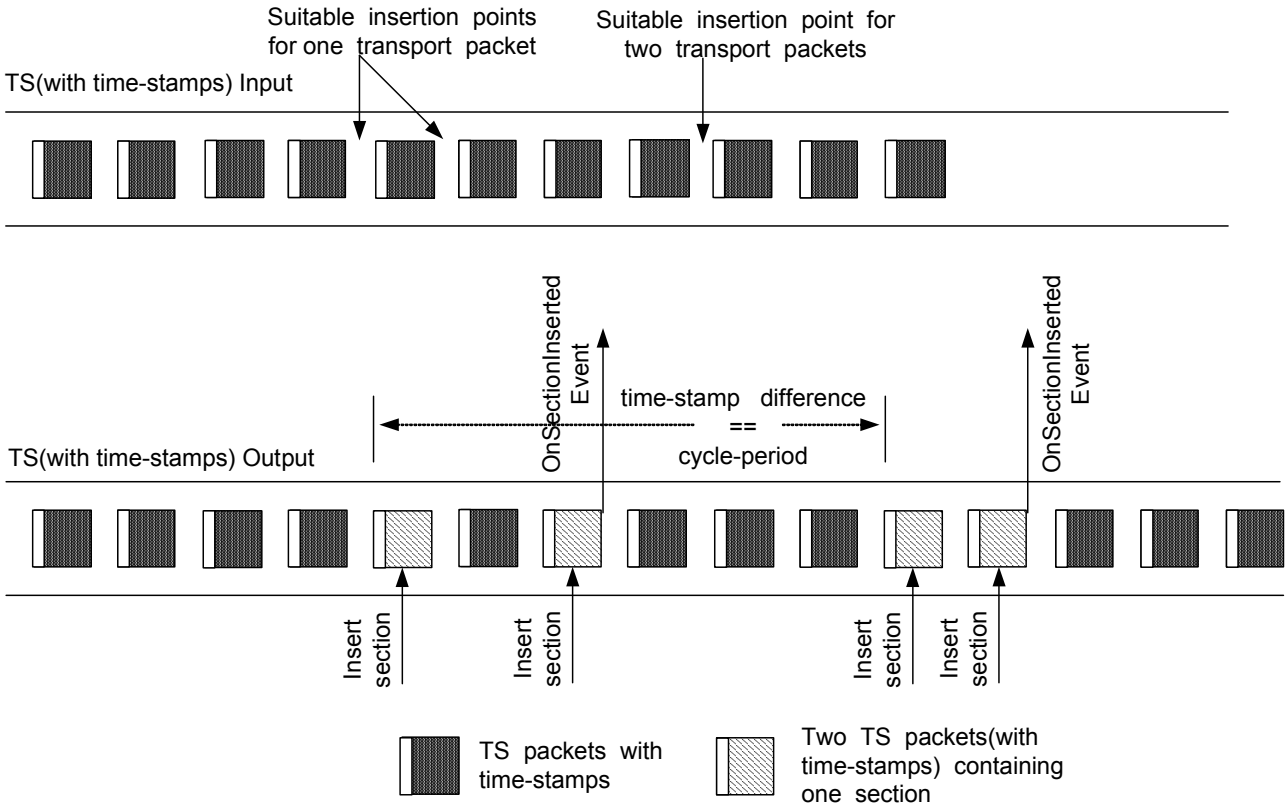


Figure 92 — Two times section data insertion into TS_{ts}

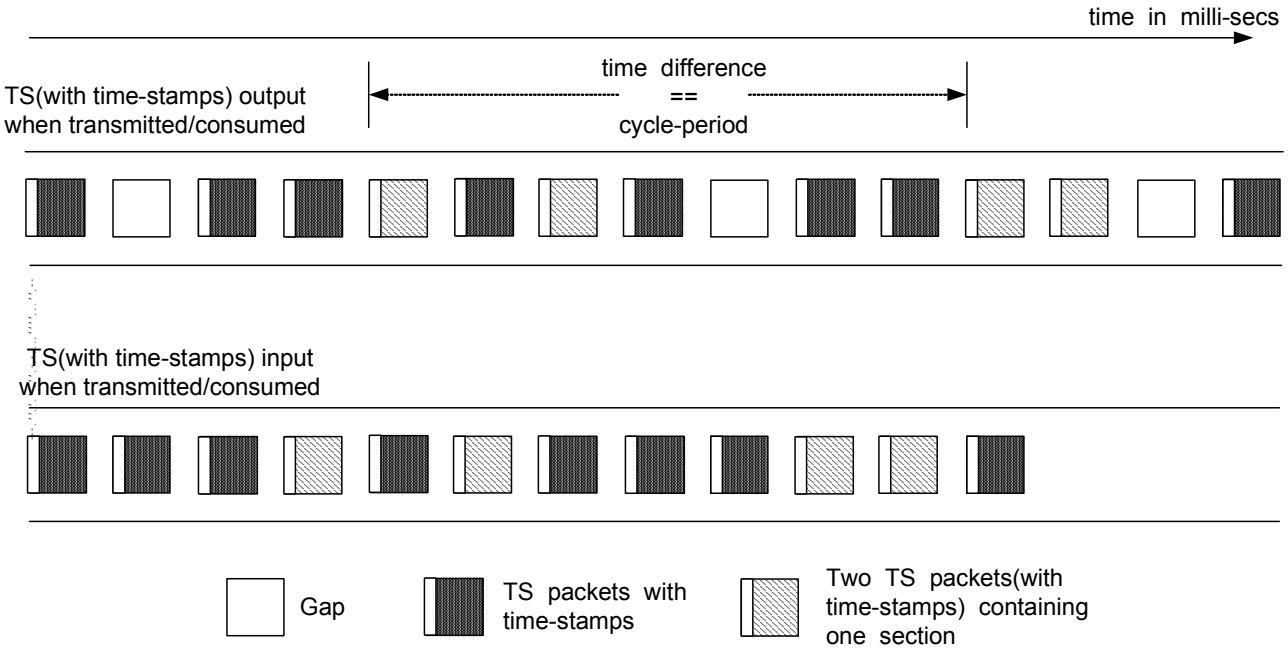


Figure 93 — TS_{ts} output by TS Mux being transmitted and/or consumed using the packet

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

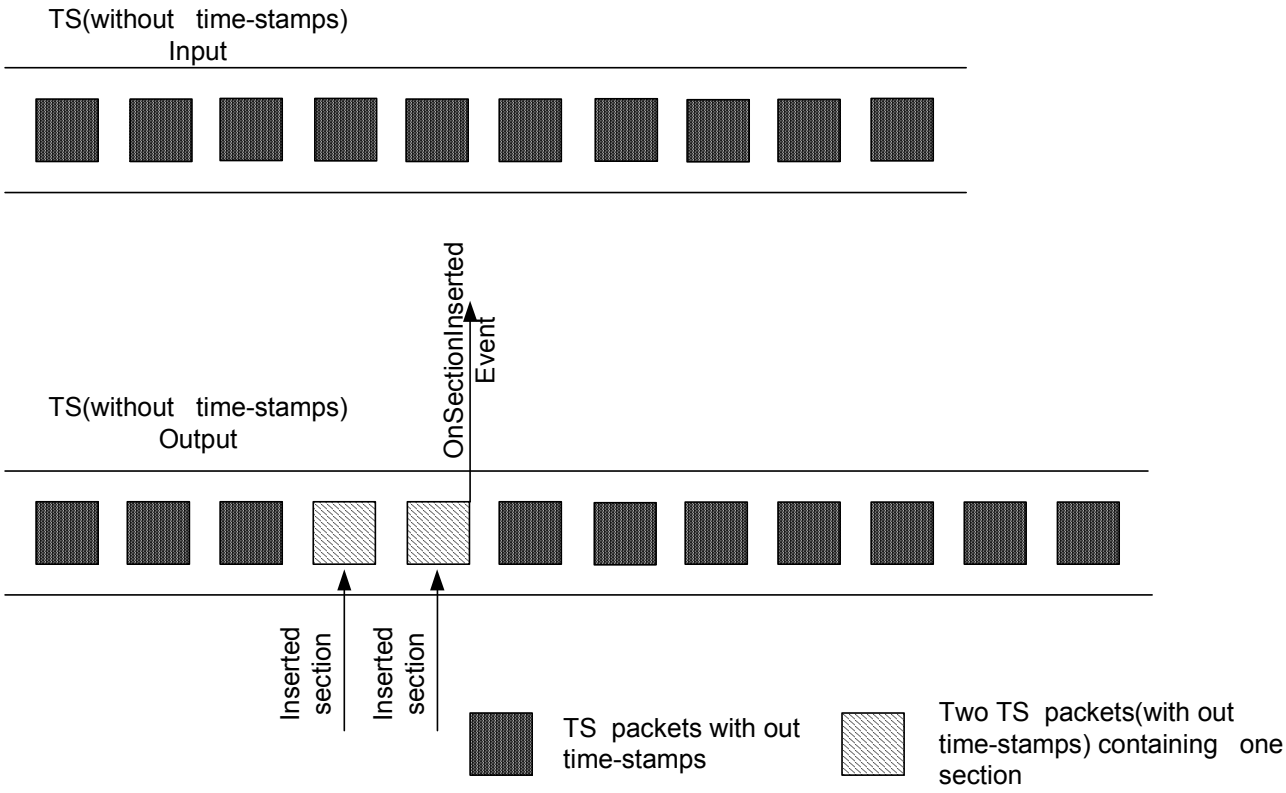


Figure 94 — One time section data insertion into TS

8.1.14.2 Types & Constants

8.1.14.2.1 Public Types & Constants

8.1.14.2.1.1 uhTsMux_Notification_t

Signature

```
typedef enum _uhTsMux_Notification_t {
    uhTsMux_SubscriptionChanged   = 0x00000001,
    uhTsMux_SectionInserted       = 0x00000002,
    uhTsMux_SectionInsertStopped  = 0x00000004
} uhTsMux_Notification_t, *puhTsMux_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhITsMuxNtf`. The values can be used to individually turn on and off notifications.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 870

Name	Description
uhTsMux_SubscriptionChanged	Represents OnSubscriptionChanged
uhTsMux_SectionInserted	Represents OnSectionInserted
uhTsMux_SectionInsertStopped	Represents OnSectionInsertStopped

8.1.14.2.1.2 uhTsMux_Notifications_t

Signature

```
typedef Int32 uhTsMux_Notifications_t, *puhTsMux_Notifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uhTsMux_Notification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.14.2.1.3 uhTsMux_AllNotifications

Signature

```
const uhTsMux_Notifications_t uhTsMux_AllNotifications = 0x00000007;
```

Qualifiers

None.

Description

Defines the set of all values of type uhTsMux_Notification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.14.2.2 Model Types & Constants

8.1.14.2.2.1 InsertableSection

Signature

```
entity InsertableSection {
    const UInt32        sectionId;
    const UInt16        pid;
    const UInt16        sectionSize;
    const UInt8*        pSection;
    const Bool          unlimitedRepeat;
    const UInt32        repeatNumber;
    const UInt32        cyclePeriod;
    Bool                started;
}
```

Qualifiers
None.

Description
Entities of this type represent a section insertable by the client into the TS/TS_{ts}. An insertable section is modeled with a unique identifier, PID value of the transport packet(s) which contain this section, its size and data pointer, one-time insertion or more, cycle-period between two insertions and whether the insertion has started or not. The `repeatNumber` is applicable only if `unlimitedRepeat` is `False`.

Independent Attributes

Table 871

Name	Description
`sectionId`	Identifier which uniquely identifies the insertable section.
`pid`	Identifier which uniquely identifies the transport packet in which the insertable section will be contained.
`sectionSize`	Size of the insertable section.
`pSection`	Pointer to the data area containing the insertable section data.
`unlimitedRepeat`	Indicates whether the section needs to be inserted indefinitely or not.
`repeatNumber`	Indicates the number of times the section needs to be inserted if `unlimitedRepeat` is `False`.
`cyclePeriod`	The cycle-period (time-gap) to be maintained between two insertions if `unlimitedRepeat` is `True` or `repeatNumber > 0`.
`started`	Indicates whether the section insertion has started or not.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.14.3 Logical Component

8.1.14.3.1 Interface-Role Model

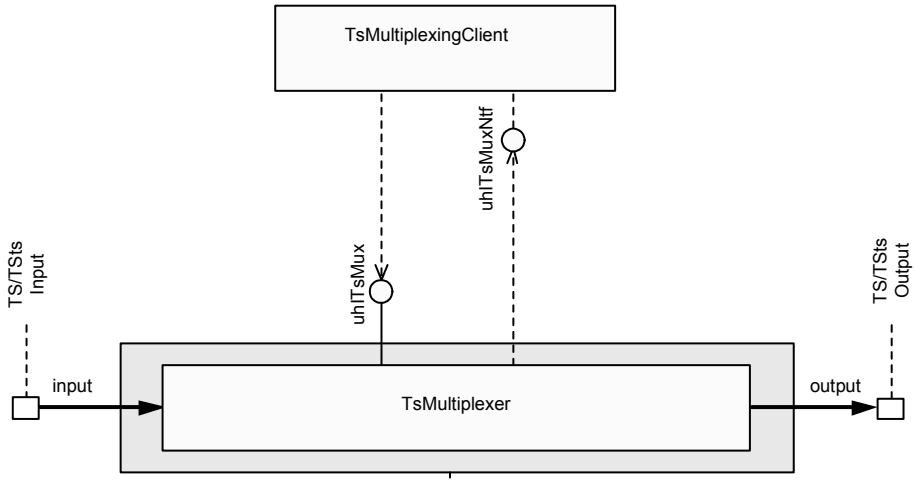


Figure 95 — Interface Role Model(TS Multiplexing)

Figure 95 depicts the interface-role model of the TS Mux logical component. It shows the control and notification interfaces and the roles involved with the TS Mux logical component. The Multiplexing role is represented by `TsMultiplexer` and the client of the TS Mux logical component is represented by `TsMultiplexingClient`.

The client can control the timed insertion of sections using the control interface `uhITsMux` provided by the `TsMultiplexer` role. The control interface can also be used to subscribe/unsubscribe to notifications of events related to section insertion. A client, represented by the `TsMultiplexingClient` role, can optionally provide a call-back interface `uhITsMuxNtf`. This interface contains functions that will be called by the `TsMultiplexer` role to notify multiplexing related events.

8.1.14.3.2 Diversity

8.1.14.3.2.1 Provided Interfaces

Table 872

Role	Interface	Presence
`TsMultiplexer`	`uhITsMux`	mandatory
`TsMultiplexingClient`	`uhITsMuxNtf`	optional

8.1.14.3.2.2 Configurable Items

Table 873

Role	Attribute
`TsMultiplexer`	`maxNrSections`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.14.3.2.3 Constraints

```
maxNrSections  ≥ 1
```

8.1.14.3.3 Instantiation

8.1.14.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 874

Type	Object	Multiplicity
TsMultiplexer	mux	1

8.1.14.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
mux.nrSections == 0
```

8.1.14.3.4 Execution Constraints

The interfaces provided by the roles of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier “thread-safe”.

8.1.14.4 Roles

8.1.14.4.1 TsMultiplexer

Signature

```
role TsMultiplexer {
    const UInt32                maxNrSections;
    setof{InsertableSection}    sections;
    UInt32                      nrSections;
}
```

Qualifiers

— root

Description

A TsMultiplexer object provides facilities to insert a number of independent sections of different lengths into the incoming TS/TS_{ts}. Each section can be inserted one or more times at appropriate locations in the TS/TS_{ts} according to the client specified time interval.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 875

Name	Description
`maxNrSections`	The maximum number of sections that can be requested by the client for insertion into the input TS/TS_{ts}.
`sections`	The set of sections that can be, or are, inserted in the input TS/TS_{ts}.

Dependent Attributes

Table 876

Name	Description
`nrSections`	The number of elements in `sections`.

Invariants

```
nrSections ≤ maxNrSections
```

Instantiation

A `TsMultiplexer` is always instantiated as part of the instantiation of the TS Mux logical component; see subclause 8.1.14.3.3.1.

Streaming Behavior

The `TsMultiplexer` continually monitors the incoming TS/TS_{ts} to find the appropriate insertion points in the stream. It then uses these insertion points to insert TS packetized sections. The `TsMultiplexer` inserts those sections into the incoming TS/TS_{ts} that are in its set of insertable sections (`sections`) and that have their `started` attribute equal to `True`. The `TsMultiplexer` inserts each section based on the section's attributes.

Active Behavior

The `TsMultiplexer` has standard notification behavior (see `uhITsMuxNtf` for the events that can be notified) and the following event driven behavior:

Table 877

Event	Action

A section has been inserted into the TS/TS_{ts}	— Let `s` be the section in `sections` that has been inserted.
	— if (not `s.unlimitedRepeat`)
	  — `s.repeatNumber = s.repeatNumber - 1`
	— notify `OnSectionInserted(*)`
	— if ((not `s.unlimitedRepeat`) and (`s.repeatNumber == 0`))
	  — `s.started = False`
	  — notify `OnSectionInsertStopped`(*)

8.1.14.4.2 TsMultiplexingClient

Signature

```
role TsMultiplexingClient {}
```

Qualifiers

— actor

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
A `TsMultiplexingClient` represents a user of the interface `uhITsMux`.

8.1.14.5 Interfaces

8.1.14.5.1 uhITsMux

Qualifiers
None

Description
This interface of a TsMultiplexer provides facilities to add a section into the section insertion list, start/stop the insertion of a section and remove a section from the section insertion list. It also provides a `Subscribe` and an `Unsubscribe` function for notifications of events related to section insertion.

Interface ID
`uuid( 6B759ECB-1A61-48df-9605-D6B71B4E0D49 )`

8.1.14.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhITsMuxNtf             *pINotify,
    [in] UInt32                  cookie,
    [in] uhTsMux_Notifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.14.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhITsMuxNtf             *pINotify,
    [in] UInt32                  cookie,
    [in] uhTsMux_Notifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.14.5.1.3 GetMaxNrSections

Signature
```
uhErrorCode_t GetMaxNrSections (
    [out] pUInt32 pMaxNrSections );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Returns the maximum number of sections that can be allocated for insertion.

Parameters

Table 878

Name	Description
pMaxNrSections	Output parameter returning the maximum number of sections supported by the TS Multiplexing logical component.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pMaxNrSections == mux.maxNrSections

8.1.14.5.1.4 GetAvailableNrSections

Signature
```
uhErrorCode_t GetAvailableNrSections (
    [out] pUInt32 pAvailableNrSections );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the available number of sections that can be allocated for insertion.

Parameters

Table 879

Name	Description
pAvailableNrSections	Output parameter returning the number of available sections.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
True

Action
None.

Postcondition
* pAvailableNrSections == mux.maxNrSections – mux.nrSections

8.1.14.5.1.5 AddSectionToInsert

Signature
```
uhErrorCode_t AddSectionToInsert (
    [in]  UInt16                        pid,
    [in]  UInt16                        sectionSize,
    [in, size_is(sectionSize)]   const pUInt8  pSection,
    [in]  Bool                          unlimitedRepeat,
    [in]  UInt32                        repeatNumber,
    [in]  UInt32                        cyclePeriod,
    [out] pUInt32                       pSectionId );
```

Qualifiers
— synchronous

Description
Adds a section to the set of sections to be inserted into the TS/TS_{ts} Input.

Parameters

Table 880

Name	Description
pid	The PID value that must be used for the insertion of the section.
sectionSize	The size of the section to be inserted.
pSection	Pointer to the section data to be inserted.
unlimitedRepeat	Flag indicating whether the section must be inserted an unlimited number of times.
repeatNumber	The number of times the section needs to be inserted. Parameter has no meaning if unlimitedRepeat==True.
cyclePeriod	The time in milliseconds between two insertions of the added section.
pSectionId	The identification of the section that has been added.

Return Values
Standard.

Precondition
— mux.nrSections < mux.maxNrSections
— 0x0000 <= pid <=0x1FFF
— sectionSize <= 4096

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
- Create a new `InsertableSection s`, such that
  - `s.pid`, `s.sectionSize`, `s.unlimitedRepeat`, `s.repeatNumber`, `s.CyclePeriod` are initialized with the corresponding input parameters.
  - The data contained in `pSection` is copied to `s.pSection`.
  - `s.started == False`
  - `s.sectionId` is a unique number such that there is no `sExisting` in `mux.sections` with `sExisting.sectionId == s.sectionId`
- Add `s` to `mux.sections`.

Postcondition
- The section `s` identified by `sectionId` has been added to `mux.sections`.
- `*pSectionId == s.sectionId`

Remarks
- The section insertion starts only after the client starts the insertion, a TS/TS_{ts} Input is available and a suitable insertion point is found. The `cyclePeriod` specified by the client for repeated section insertion is maintained by the TS Mux only in the best-effort mode and is not guaranteed to be the same for all the insertions of a particular section.

8.1.14.5.1.6 DeleteSectionToInsert

Signature
```
uhErrorCode_t DeleteSectionToInsert (
    [in] UInt32  sectionId );
```

Qualifiers
- synchronous

Description
Deletes a section from the section insertion list.

Parameters

Table 881

Name	Description
`sectionId`	The identification of the section to be deleted.

Return Values
Standard.

Precondition
- `sectionId` identifies a section `s` in `mux.sections` such that `s.sectionId == sectionId`.
- `s.started == False`

Action
- Modify `mux.sections`.

Postcondition
- The section `s` identified by `sectionId` has been removed from `mux.sections`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks
A section can be deleted only after the section insertion process has been stopped. The client, before deleting a section, can either use the stopped notification event `OnSectionInsertStopped` for the insertion process of a particular section that was requested to be stopped or poll for the stopped status by using the `GetSectionInsertionStatus()` function call.

8.1.14.5.1.7 StartSectionInsertion

Signature
```
uhErrorCode_t StartSectionInsertion (
    [in] UInt32  sectionId  );
```

Qualifiers
— asynchronous

Description
Start the section insertion process for this section.

Parameters

Table 882

Name	Description
`sectionId`	The identification of the section for which the insertion process needs to be started.

Return Values
Standard.

Precondition
— `sectionId` identifies a section `s` in `mux.sections` such that `s.sectionId == sectionId.`

Action
— Modify `s.started.`

Postcondition
`s.started == True`

Asynchronous Action
— Refer to the ‘Active Behaviour’ specified in the subclause 7.2.3.4.1 for the actions carried out asynchronously.

Asynchronous Postcondition
— Refer to the ‘Active Behaviour’ specified in the subclause 7.2.3.4.1 for the asynchronous actions and the corresponding postconditions.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.14.5.1.8 StopSectionInsertion

Signature
```
uhErrorCode_t StopSectionInsertion (
    [in] UInt32  sectionId );
```

Qualifiers
— asynchronous

Description
Stop the section insertion process for this section.

Parameters

Table 883

Name	Description
`sectionId`	The identification of the section for which the insertion process needs to be stopped.

Return Values
Standard.

Precondition
— `sectionId` identifies a section `s` in `mux.sections` such that `s.sectionId == sectionId`.

Asynchronous Action
— Let `s` be the section in `mux.sections` whose insertion process needs to be stopped.

— Modify `s.started`.

— Notify `OnSectionInsertStopped( * )`.

Asynchronous Postcondition
```
s.started == False
```

8.1.14.5.1.9 GetSectionInsertionStatus

Signature
```
uhErrorCode_t GetSectionInsertionStatus (
    [in]  UInt32   sectionId,
    [out] pBool    pSectionInsertionStatus );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current insertion status of the section.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 884

Name	Description
`sectionId`	The identification of the section for which the insertion process status is requested.
`pSectionInsertionStatus`	Output parameter returning the section insertion status. `True` if the insertion process of this section is ongoing, `False` if not.

Return Values
Standard.

Precondition
— `sectionId` identifies a section `s` in `mux.sections` such that `s.sectionId == sectionId`.

Action
None.

Postcondition
`*pSectionInsertionStatus == s.starteduhI`TsMuxNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a `TsMultiplexer`. The subscribe function of the `uhITsMux` interface can be used to subscribe to these events.

Interface ID
`uuid( CAFCB8DB-DE00-4250-99F1-73603DC11A16 )`

8.1.14.5.1.10 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhTsMux_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.14.5.1.11 OnSectionInserted

Signature
```
[async] Void OnSectionInserted (
```

Copyright International Organization for Standardization © ISO/IEC 2007 All rights reserved
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    [in] UInt32  cookie,
    [in] UInt32  sectionId );
```

Qualifiers
None.

Description
It is called when the `TsMultiplexer` successfully inserts a section into the TS/TS_{ts} Input.

Parameters

Table 885

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`sectionId`	The identification of the section that was inserted.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.14.5.1.12 OnSectionInsertStopped

Signature

```
[async] Void OnSectionInsertStopped (
    [in] UInt32  cookie,
    [in] UInt32  sectionId );
```

Qualifiers
None.

Description
It is called when the `TsMultiplexer` stops the section insertion either because of the client's request or the client requested number of insertions is completed.

Parameters

Table 886

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`sectionId`	The identification of the section whose insertion has been stopped.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.15 UhlTuning

8.1.15.1 Concepts

8.1.15.1.1 Introduction

A Tuning logical component receives Radio Frequency (RF) signals via an external input (i.e. an antenna, cable or satellite dish). It can perform many operations like tuning, fine-tuning and optionally Automatic Frequency Control (AFC) and searching on the input signal.

8.1.15.1.2 Tuning and searching

Tuning is extracting one modulated signal from the wide RF spectrum that is received and providing it at the output at an Intermediate Frequency (IF) so that it can be demodulated, split and processed by other logical components further on in the streaming path. Note that in our definition, a signal does not imply meaningful content, it simply means that we detect something else than noise on the input.

In order to get a signal with meaningful content, we need to tune to the correct frequency. A frequency can be set directly but also indirectly using a channel ID representing a frequency. In most cases to know the frequencies with meaningful content, we have to search the frequency band for them. With respect to searching, various algorithms are used depending on the hardware used and the region of the world the tuner has to operate in.

In order to start a search the following information is required: the start position, the search direction and the end position. For analog systems searching is also dependent on the modulation of the signal. It is possible to do a frequency search, which searches a given frequency range or a channel search, which allows a more optimal way of searching. After a search has started, notifications will indicate when the search has ended. A search will end when a signal has been found, the stop criteria was met or the search has been stopped by the client. If a search has ended because a signal has been found, it is possible to check more accurately if that signal represents meaningful content by inspecting properties of other logical components further on in the streaming path. Typically, these other logical components will also notify these properties.

8.1.15.1.3 Frequencies and channels

A frequency denotes an exact position in a frequency spectrum. A more abstract positioning in a frequency spectrum is possible by means of channels. A channel represents a frequency including a predefined lower and upper bound for that frequency. A channel is typically identified by an ID (a channel number). The mapping of channel numbers to frequencies is specified by means of a channel table. Channel tables can be different per country or region. A frequency can be set to any value (in Hertz) between a given minimum and maximum frequency depending on the capabilities of the tuner. Channel numbers also can be set between a

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

given minimum and maximum but do not have to form a consecutive list. Hence, a channel table can contain gaps and does not need to cover the complete frequency band depending on region or country.

8.1.15.1.4 Fine-tuning

Fine-tuning is used to locate the carrier frequency of a signal in a predefined window around a given frequency as fast as possible. A fine-tuning action is executed once and is finished after the carrier frequency has been found. It is possible to fine-tuning when setting a channel as well as a frequency.

8.1.15.1.5 AFC

The AFC is used to continuously monitor the carrier frequency of a signal tuned to. If the carrier frequency drifts, the tuning is adjusted. This way it is possible to follow/track drifting transmitters such as a VCR in order to keep optimal signal reception. The AFC can only track the frequency within a predefined window.

8.1.15.2 Types & Constants

8.1.15.2.1 Public Types & Constants

8.1.15.2.1.1 Error Codes

None.

8.1.15.2.1.2 uhTuning_Notification_t

Signature

```
typedef enum _uhTuning_Notification_t {
    uhTuning_SubscriptionChanged = 0x00000001,
    uhTuning_Tuned               = 0x00000002,
    uhTuning_Aborted             = 0x00000004,
    uhTuning_SignalPresence      = 0x00000008,
    uhTuning_AfcFreqChanged      = 0x00000010,
    uhTuning_AfcLimitReached     = 0x00000020
} uhTuning_Notification_t, *puhTuning_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the notification functions of the interface `uhITuningNtf`. They can be used to individually turn on and off notifications.

Values

Table 887

Name	Description
`uhTuning_SubscriptionChanged`	Represents `OnSubscriptionChanged`
`uhTuning_Tuned`	Represents `OnActEnabledChanged`
`uhTuning_Aborted`	Represents `OnAborted`
`uhTuning_SignalPresence`	Represents `OnSignalPresence`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTuning_AfcFreqChanged	Represents OnAfcFreqChanged
uhTuning_AfcLimitReached	Represents OnAfcLimitReached

8.1.15.2.1.3 uhTuning_Notifications_t

Signature
```
typedef Int32 uhTuning_Notifications_t, *puhTuning_Notifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing values of type uhTuning_Notification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.15.2.1.4 uhTuning_AllNotifications

Signature
```
const uhTuning_Notifications_t uhTuning_AllNotifications = 0x0000003F;
```

Qualifiers
None.

Description
The set of all notifications of uhTuning_Notification_t.This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.1.15.2.1.5 uhTuning_SearchNotification_t

Signature
```
typedef enum _uhTuning_SearchNotification_t {
    uhTuning_SearchSubscriptionChanged = 0x00000001,
    uhTuning_SignalDetected            = 0x00000002,
    uhTuning_Stopped                   = 0x00000004,
    uhTuning_EndPosReached             = 0x00000008,
    uhTuning_FreqChanged               = 0x00000010,
    uhTuning_ChannelChanged            = 0x00000020
} uhTuning_SearchNotification_t, *puhTuning_SearchNotification_t;
```

Qualifiers
— enum-element

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent the notification functions in the interface uhIAdrcNtf. They can be used to individually turn on and off notifications.

Values

Table 888

Name	Description
uhTuning_SearchSubscriptionChanged	Represents OnSubscriptionChanged
uhTuning_SignalDetected	Represents OnSignalDetected.
uhTuning_Stopped	Represents OnStopped
uhTuning_EndPosReached	Represents OnEndPosReached.
uhTuning_FreqChanged	Represents OnFreqChanged.
uhTuning_ChannelChanged	Represents OnChannelChanged

8.1.15.2.1.6 uhTuning_SearchNotifications_t

Signature
```
typedef Int32 uhTuning_SearchNotifications_t, *puhTuning_SearchNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of uhTuning_SearchNotification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.1.15.2.1.7 uhTuning_SearchAllNotifications

Signature
```
const uhTuning_SearchNotifications_t
      uhTuning_SearchAllNotifications = 0x0000003F;
```

Qualifiers
None.

Description
The set of all notifications of uhTuning_SearchNotification_t.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.15.2.1.8 uhTuning_ChanTableType_t

Signature

```
typedef enum _uhTuning_ChanTableType_t {
    uhAnaVdec_VideoCodingCVBS  = 1,
    uhAnaVdec_VideoCodingYC    = 2
} uhTuning_ChanTableType_t, *puhTuning_ChanTableType_t;
```

Qualifiers
None.

Description
Possible channel table types based on input type. This information might be used for optimization purposes (i.e. the search algorithm used when searching).

Values

Table 889

Name	Description
uhTuning_ChanTableTypeAntenna	The channel table type for antenna input.
uhTuning_ChanTableTypeCable	The channel table type for cable input.

8.1.15.2.1.9 uhTuning_AfcValue_t

Signature

```
typedef enum _uhTuning_AfcValue_t {
    uhTuning_AfcValueInvalid = -2,
    uhTuning_AfcValueUnknown = -1,
    uhTuning_FarBelowIf      =  1,
    uhTuning_BelowIf         =  2,
    uhTuning_AboveIf         =  3,
    uhTuning_FarAboveIf      =  4
} uhTuning_AfcValue_t, *puhTuning_AfcValue_t;
```

Qualifiers
None.

Description
Values of this type give an indication on how much the current frequency deviates from the IF due to AFC.

Values

Table 890

Name	Description
uhTuning_AfcValueInvalid	The AFC value is temporarily not available.
uhTuning_AfcValueUnknown	The AFC value is available, yet could not be determined.
uhTuning_FarBelowIf	The current frequency is outside the AFC window and below IF.
uhTuning_BelowIf	The current frequency is inside the AFC window and below IF.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTuning_AboveIf	The current frequency is inside the AFC window and above IF.
uhTuning_FarAboveIf	The current frequency is outside the AFC window and above IF.

8.1.15.2.1.10 uhTuning_ChanTableData_t

Signature

```
typedef struct _uhTuning_ChanTableData_t {
    UInt32 firstChan;
    UInt32 lowerFreq;
    UInt32 nrofChans;
    UInt32 freqWidth;
    UInt32 freqOffset;
} uhTuning_ChanTableData_t, *puhTuning_ChanTableData_t;
```

Qualifiers
None.

Description
This type defines the layout of a single data entry of a channel table. A complete channel table will consist of an array of these structures. By using this layout, channel tables can be set in a compact and efficient way.

Example:

Table 891

firstChan	lowerFreq	nrofChans	freqWidth	freqOffset
2	70,250,000	1	4,000,000	3,000,000
3	52,250,000	2	6,000,000	3,000,000
9	90,250,000	4	6,000,000	3,000,000

This table is the short form of the following table:

Table 892

channel	lowerFreq	upperFreq	carrierFreq
2	70,250,000	74,250,000	73,250,000
3	52,250,000	58,250,000	55,250,000
4	58,250,000	64,250,000	61,250,000
9	90,250,000	96,250,000	93,250,000
10	96,250,000	102,250,000	99,250,000
11	102,250,000	108,250,000	105,250,000
12	108,250,000	114,250,000	111,250,000

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Fields

Table 893

Name	Description
`firstChan`	The first channel in the sequence of channels represented by this data entry.
`lowerFreq`	The lower frequency in Hz for the sequence of channels.
`nrofChans`	The number of channels in the sequence of channels. So `firstChan + nrofChans - 1 == lastChan`.
`freqWidth`	The frequency width in Hz for each channel in the sequence of channels. So `lowerFreq + freqWidth == upperFreq` for first channel in the sequence of channels (all in Hz).
`freqOffset`	The offset to reach the carrier frequency in Hz for each channel in the sequence of channels. So `lowerFreq + freqOffset == carrierFreq` for the first channel in the sequence of channels (all in Hz).

Constraints

— Smallest frequency used in table **>=** `Aavd.minAmp`
— Largest frequency used in table **<=** `tuner.maxFreq`

8.1.15.2.1.11 uhTuning_SearchModeFlag_t

Signature

```
typedef enum _uhTuning_SearchModeFlag_t {
    uhTuning_ModulationNone = 0x00000001,
    uhTuning_ModulationNeg  = 0x00000002,
    uhTuning_ModulationPos  = 0x00000004,
    uhTuning_ChannelBased   = 0x00000008,
    uhTuning_FreqBased      = 0x00000010,
    uhTuning_DirectionUp    = 0x00000020,
    uhTuning_DirectionDown  = 0x00000040,
    uhTuning_CurrentIncl    = 0x00000080,
    uhTuning_CurrentExcl    = 0x00000100,
    uhTuning_FineTuneIncl   = 0x00000200,
    uhTuning_FineTuneExcl   = 0x00000400
} uhTuning_SearchModeFlag_t, *puhTuning_SearchModeFlag_t;
```

Qualifiers

— enum-element

Description

This type represents the possible search mode flags.

Values

Table 894

Name	Description
`uhTuning_ModulationNone`	Modulation type unspecified
`uhTuning_ModulationNeg`	Search for negative modulated signals.
`uhTuning_ModulationPos`	Search for positive modulated signals.
`uhTuning_ChannelBased`	Search based on channels using a channel table.
`uhTuning_FreqBased`	Search based on frequency.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhTuning_DirectionUp	Search in upward direction.
uhTuning_DirectionDown	Search in downward direction.
uhTuning_CurrentIncl	Search with current position included.
uhTuning_CurrentExcl	Search with current position excluded.
uhTuning_FineTuneIncl	Search including a fine-tune action.
uhTuning_FineTuneExcl	Search excluding a fine-tune action.

8.1.15.2.1.12 uhTuning_SearchMode_t

Signature

```
typedef Int32 uhTuning_SearchMode_t, *puhTuning_SearchMode_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of search mode flags that are typically constructed by logical OR-ing value of type `uhTuning_SearchModeFlag_t`.

8.1.15.2.1.13 uhTuning_ValStat_t

Signature

```
typedef enum _uhTuning_ValStat_t {
    uhTuning_ValStatInvalid = -2,
    uhTuning_ValStatUnknown = -1,
    uhTuning_ValStatKnown   =  1
} uhTuning_ValStat_t, *puhTuning_ValStat_t;
```

Qualifiers

None.

Description

This type represents the validity status of the related property/properties.

Values

Table 895

Name	Description
uhTuning_ValStatInvalid	The signal strength property is temporarily not available.
uhTuning_ValStatUnknown	The signal strength property is available, yet could not be determined.
uhTuning_ValStatKnown	The signal strength property is available and determined.

8.1.15.2.2 Model Types & Constants

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.15.3 Logical Component

8.1.15.3.1 Interface-Role Model

Figure 96 depicts the interface-role model of the Tuning logical component. It shows the control and notification interfaces and the roles involved with this logical component.

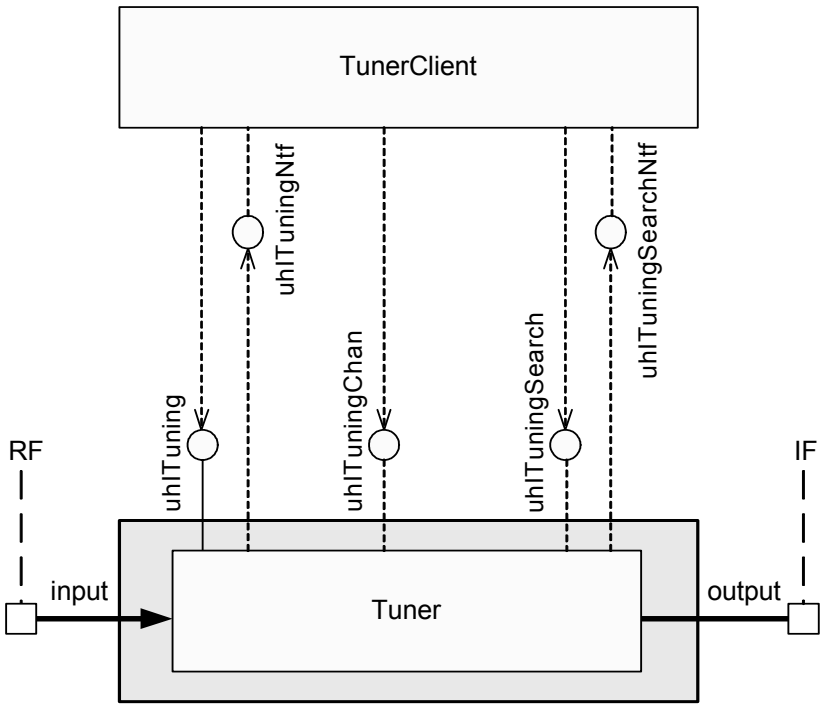


Figure 96 — Interface-Role Model

The grey box in Figure 96 denotes a Tuning logical component containing the `Tuner` role providing the mandatory (solid line) and optional (dotted line) control interfaces to the client.

The clients can optionally provide call-back interfaces to receive notifications of events in the `Tuner`.

8.1.15.3.2 Diversity

8.1.15.3.2.1 Provided Interfaces

Table 896

Role	Interface	Presence
`Tuner`	`uhITuning`	mandatory
`Tuner`	`uhITuningChan`	optional

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Role	Interface	Presence
Tuner	if (tuner.tuning == True) OR (tuner.searching == True) — tuner.actChannelStatus != uhTuning_ValStatKnown — else if (tuner.actFreq not in channel of channel table) — tuner.actChannelStatus != uhTuning_ValStatKnown — else — tuner.actChannelStatus != uhTuning_ValStatUnknown — *pChannelStatus == tuner.actChannelStatus — *pChannel == tuner.actChannel	optional
	uhITuningSearch	
TunerClient	uhITuningNtf	optional
TunerClient	uhITuningSearchNtf	optional

8.1.15.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of the logical component. For a description of the attributes see 8.1.15.4 and for how they are initialized see subclause 8.1.15.3.3.

Table 897

Role	Attribute
Tuner	minAmp
Tuner	maxFreq
Tuner	afcSupported
Tuner	maxChannelTableSize
Tuner	minChannelOffset
Tuner	maxChannelOffset
Tuner	suppSearchModes

8.1.15.3.2.3 Constraints

See the *Invariants* of the Tuner role (8.1.15.4).

8.1.15.3.3 Instantiation

8.1.15.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 898

Type	Object	Multiplicity
Tuner	tuner	1

8.1.15.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

- tuner.setFreq == tuner.minAmp
- tuner.actFreqStatus == uhTuning_ValStatInvalid
- tuner.actFreq == tuner.minAmp
- tuner.setFreqFineTune == False
- tuner.fineTune == False
- tuner.tuning == False
- tuner.presenceStatus == uhTuning_ValStatInvalid
- tuner.signalPresence == False
- tuner.enableAfc == False
- tuner.afcValue == uhTuning_AfcValueInvalid
- tuner.channelTableId == undefined
- tuner.channelTableType == undefined
- tuner.channelTableSize == 0
- tuner.channelTable == null
- tuner.setChannel == undefined
- tuner.actChannelStatus == uhTuning_ValStatInvalid
- tuner.actChannel == undefined
- tuner.channelOffset == 0
- tuner.setChannelFineTune == False
- tuner.searching == False
- tuner.stopSearch == False

8.1.15.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.15.4 Roles

8.1.15.4.1 Tuner

Signature

```
role Tuner {
    const UInt32                minAmp;
    const UInt32                maxFreq;
    UInt32                      setFreq;
    uhTuning_ValStat_t          actFreqStatus;
    UInt32                      actFreq;
    Bool                        setFreqFineTune;
    Bool                        fineTune;
    Bool                        tuning;
    uhTuning_ValStat_t          presenceStatus;
    Bool                        signalPresence;
    const Bool                  afcSupported;
    Bool                        enableAfc;
    uhTuning_AfcValue_t         afcValue;
    const UInt32                maxChannelTableSize;
    UInt32                      channelTableId;
    uhTuning_ChanTableType_t    channelTableType;
    UInt32                      channelTableSize;
    uhTuning_ChanTableData_t    channelTable[channelTableSize];
    const Int32                 minChannelOffset;
    const Int32                 maxChannelOffset;
    UInt32                      setChannel;
    uhTuning_ValStat_t          actChannelStatus;
    UInt32                      actChannel;
    Int32                       channelOffset;
    Bool                        setChannelFineTune;
    const uhTuning_SearchMode_t suppSearchModes;
    Bool                        searching;
    Bool                        stopSearch;
}
```

Qualifiers

— root

Description

A `Tuner` controls the tuning attributes of the front end.

Independent Attributes

Table 899

Name	Description
`minFreq`	The minimum frequency that can be set (in Hz).
`maxFreq`	The maximum frequency that can be set (in Hz).
`setFreq`	The frequency as it is set by the client.
`actFreqStatus`	The validity status of the actual frequency.
`actFreq`	The actual frequency tuned to.
`setFreqFineTune`	The fine tune parameter as set by the last `SetFreq` call of the client.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
fineTune	Indicates if a fine tune needs to be done (True) or not (False)
tuning	Indicates if tuning is busy (True) or not (False).
presenceStatus	The validity status of the signal presence.
signalPresence	Indicates if on the actual frequency the presence of a signal has been detected (True) or not (False)
afcSupported	Indicates if AFC control is supported (True) or not (False).
enableAfc	The setting as set by the client indicating if AFC should be enabled (True) or disabled (False).
afcValue	The AFC value indicating how much the actual frequency deviates from the IF.
maxChannelTableSize	The maximum channel table size (in table entries).
channelTableId	The identification of the current channel table.
channelTableType	The type of channel table currently in use.
channelTableSize	The size (in table entries) of the current channel table.
channelTable	The current channel table.
minChannelOffset	The minimum offset for a channel.
maxChannelOffset	The maximum offset for a channel.
setChannel	The channel set by the client.
actChannelStatus	The validity status of the actual channel.
actChannel	The actual channel.
channelOffset	The frequency offset as set by the client.
setChannelFineTune	The fine tune parameter as set by the last SetChannel call of the client.
suppSearchModes	The supported search modes.
searching	Indicates if a search is active (True) or not (False).
stopSearch	Indicates if the active search should be stopped (True) or not (False).

Invariants

```
—— maxFreq >= minAmp
—— minAmp <= setFreq <= maxFreq
—— minAmp <= actFreq <= maxFreq
—— sizeof ( channelTable ) <= maxChannelTableSize
—— minChannelOffset <= 0
—— maxChannelOffset >= 0
—— minChannelOffset <= channelOffset <= maxChannelOffset
—— channelTableSize <= maxChannelTableSize
```

Instantiation

A Tuner role is always instantiated as part of a Tuning logical component; see subclause 8.1.15.3.3.

Streaming Behavior

The Tuner role tunes to the incoming RF signal and outputs an IF signal that can be handled by a demodulator/decoder.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Active Behavior
The `Tuner` has standard notification behavior, except for the following events:

Table 900

Event	Action
The actual frequency changed due to AFC.	Notify `OnAfcFreqChanged`
The actual frequency changed and reached a limit of the AFC window due to AFC.	Notify `OnAfcLimitReached`
The actual frequency has changed during a frequency based search (progress indication).	Notify `OnFreqChanged`
The channel changed during a channel based search (progress indication).	Notify `OnChannelChanged`

8.1.15.4.2 TunerClient

Signature
`role` TunerClient `{}`

Qualifiers
— actor

Description
A TunerClient represents a user of the control interfaces offered by the Tuner. It can optionally provide the notification interfaces to receive notifications of events generated by the Tuner. No constraints apply to the behavior of the TunerClient other than those implied by Figure 96.

8.1.15.5 Interfaces

8.1.15.5.1 uhlTuning

Qualifiers
None.

Description
This interface provides control over basic tuning functionality.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( AB238C03-4170-450b-AC2A-A91334C03BD2 )`

8.1.15.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhITuningNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhTuning_Notifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.15.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhITuningNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhTuning_Notifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.15.5.1.3 GetFreqRange

Signature
```
uhErrorCode_t GetFreqRange (
    [out] pUInt32 pMinFreq,
    [out] pUInt32 pMaxFreq );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the boundaries for the frequency that can be set by client.

Parameters

Table 901

Name	Description
`pMinFreq`	Output parameter returning the minimum frequency in Hz.
`pMaxFreq`	Output parameter returning the maximum frequency in Hz.

Return Values
Standard

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None

Postcondition
— `*pMinFreq == Aavd.minAmp`
— `*pMaxFreq == tuner.maxFreq`

8.1.15.5.1.4 SetFreq

Signature
```
uhErrorCode_t SetFreq (
    [in] UInt32 freq,
    [in] Bool fineTune );
```

Qualifiers
— asynchronous

Description
This function sets the frequency. A client can request a fine tune action. If a search is in progress the search will be stopped by calling this function.

Parameters

Table 902

Name	Description
`freq`	Input parameter indicating the new frequency in Hz.
`fineTune`	Input parameter indicating whether a fine-tune action should be done (`True`) or not (`False`). Tuners not capable of fine tuning should ignore this parameter.

Return Values
Standard

Precondition
`tuner.minAmp <= freq <= tuner.maxFreq`

Action
— if `(tuner.tuning == True )`
  — Notify `OnAborted (tuner.actFreq )`
— if `(tuner.searching == True )`
  — Modify `tuner.stopSearch`
— Modify `tuner.setFreq`
— `Modify tuner.setFreqFineTune`
— `Modify tuner.fineTune`
— `Modify tuner.tuning`

Postcondition
— `tuner.stopSearch == True`
— `tuner.setFreq == freq`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— tuner.setFreqFineTune == fineTune
— tuner.fineTune == fineTune
— tuner.tuning == True

Asynchronous Action
— Execute tune action
— if (tuner.fineTune)
  — Execute fine tune action
— Modify tuner.actFreq
— Modify tuner.fineTune
— Modify tuner.tuning
— Modify tuner.signalPresence
— Notify OnTuned(tuner.actFreq)
— if (signal found)
  — Notify OnSignalPresence(tuner.actFreq)

Asynchronous Postcondition
— if (tuner.fineTune)
  — tuner.actFreq == round(finetune(freq))
— else
  — tuner.actFreq == round(tune(freq))
— tuner.fineTune == False
— tuner.tuning == False
— if (signal found)
  — tuner.signalPresence == True
— else
  — tuner.signalPresence == False

Remarks
— The actual frequency (tuner.actFreq) is rounded to the nearest frequency that is supported by the implementation.

8.1.15.5.1.5 GetFreq

Signature
```
uhErrorCode_t GetFreq (
    [out] pUInt32 pFreq,
    [out] pBool pFineTune );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the frequency parameters as they where set by the last SetFreq function call.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 903

Name	Description
pFreq	Output parameter returning the frequency in Hz that was set.
pFineTune	Output parameter returning the fine tune parameter that was as set.

Return Values
Standard

Precondition
True

Action
None

Postcondition
— *pFreq == Aavd.setFreq
— *pFineTune == tuner.setFreqFineTune

8.1.15.5.1.6 GetActualFreq

Signature

```
uhErrorCode_t GetActualFreq (
    [out] pBool   pTuned,
    [out] puhTuning_ValStat_t pActFreqStatus,
    [out] pUInt32 pFreq );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the actual frequency tuned to. The actual frequency depends on rounding, fine tuning and AFC.

Parameters

Table 904

Name	Description
pTuned	Output parameter indicating whether a tune action is still busy (True) or not (False). If the tune action is still busy, the actual frequency has not settled yet.
pActFreqStatus	Output parameter returning the validity status of the actual frequency.
pFreq	Output parameter returning the actual frequency in Hz. If pTuned equals True, the frequency value has settled (is not in a transitory state). If pTuned equals False, the returned frequency value has not settled yet. If pTuned equals True the actual frequency may still vary due to of AFC.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard

Precondition
`True`

Action
None

Postcondition
- `*pActFreqStatus == Aavd.actFreqStatus`
- `*pFreq == tuner.actFreq`
- `if (tuner.tuning == True ) OR (tuner.searching == True )`
  - `*pTuned == False`
- `else`
  - `*pTuned == True`

8.1.15.5.1.7 GetSignalPresence

Signature
```
uhErrorCode_t GetSignalPresence (
    [out] puhTuning_ValStat_t pPresenceStatus,
    [out] pBool pPresence );
```

Qualifiers
- synchronous
- thread-safe

Description
This functions returns whether the presence of a signal was detected on the frequency currently tuned to.

Parameters

Table 905

Name	Description
`pPresenceStatus`	Output parameter returning the validity status of the signal presence.
`pPresence`	Output parameter indicating whether the presence of a signal was detected (`True`) or not (`False`).

Return Values
Standard

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None

Postcondition
— `*pPresenceStatus ==` Aavd.`presenceStatus`
— `*pPresence == tuner.signalPresence`

8.1.15.5.1.8 GetAfcSupported

Signature
```
uhErrorCode_t GetAfcSupported ( [out] pBool pSupported );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns if AFC control is supported.

Parameters

Table 906

Name	Description
`pSupported`	Output parameter indicating if AFC control is supported (`True`) or not (`False`).

Return Values
Standard

Precondition
`True`

Action
None

Postcondition
`*pSupported == tuner.afcSupported`

8.1.15.5.1.9 EnableAfc

Signature
```
uhErrorCode_t EnableAfc ( [in] Bool enable );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is used by the client to indicate that AFC should be enabled (`True`) or disabled (`False`). If AFC is enabled by the client, the platform will activate the AFC but might decide to deactivate it temporarily if required (i.e. when fine tuning or searching etc).

The deactivation of the AFC is not directly observable by the client. When the AFC is active the actual frequency tuned to will possibly be influenced.

Parameters

Table 907

Name	Description
`enable`	Input parameter indicating if AFC has to be enabled (`True`) or disabled (`False`).

Return Values
Standard

Precondition
Aavd`.afcSupported == True`

Action
— Modify `tuner.enableAfc`

Postcondition
`tuner.enableAfc == enable`

8.1.15.5.1.10 GetAfcEnabled

Signature
```
uhErrorCode_t GetAfcEnabled ( [out] pBool pEnable );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the AFC enable setting as set by the client. If AFC is enabled by the client, the platform will activate the AFC but might decide to deactivate it temporarily if required (i.e. when fine tuning or searching etc). The deactivation of the AFC is not directly observable by the client.

Parameters

Table 908

Name	Description
`pEnable`	Output parameter indicating if AFC is enabled (`True`) or disabled (`False`).

Return Values
Standard

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
Aavd.`afcSupported == True`

Action
None

Postcondition
`*pEnable == tuner.enableAfc`

8.1.15.5.1.11 GetAfcValue

Signature
`uhErrorCode_t GetAfcValue ( [out] puhTuning_AfcValue_t pValue );`

Qualifiers
— synchronous

— thread-safe

Description
This function returns the AFC value. This value indicates how much the current frequency deviates from the IF due to AFC.

Parameters

Table 909

Name	Description
`pValue`	Output parameter returning the AFC value. If AFC is disabled this function will return `uhTuning_AfcValueUnknown`.

Return Values
Standard

Precondition
Aavd.`afcSupported == True`

Action
None

Postcondition
`*pValue == tuner.afcValue`

8.1.15.5.2 uhITuningNtf

Qualifiers
— callback

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface can be provided by a client that wants to receive notifications with respect to basic tuning.

Interface ID
```
uuid( D2AD0211-B451-455b-89EA-F7B3D3E2C438 )
```

8.1.15.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhTuning_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.15.5.2.2 OnTuned

Signature
```
[async] Void OnTuned (
    [in] UInt32 cookie,
    [in] uhTuning_ValStat_t actFreqStatus,
    [in] UInt32 actFreq );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the tune action has finished.

Parameters

Table 910

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`actFreqStatus`	The validity status of the notified actual frequency.
`actFreq`	The actual frequency tuned to.

Return Values
None.

Precondition
`True`

Action
— Any callback compliant action

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
True

8.1.15.5.2.3 OnAborted

Signature
```
[async] Void OnAborted (
    [in] UInt32 cookie,
    [in] UInt32 actFreq );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the tune action has been aborted.

Parameters

Table 911

Name	Description
cookie	The value that was passed by the client at subscription time.
actFreq	The actual frequency tuned to.

Return Values
None.

Precondition
True

Action
— Any callback compliant action

Postcondition
True

8.1.15.5.2.4 OnSignalPresence

Signature
```
[async] Void OnSignalPresence (
    [in] UInt32 cookie,
    [in] uhTuning_ValStat_t status,
    [in] UInt32 actFreq,
    [in] Bool   presence );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function notifies subscribed clients whether or not the presence of a signal has been detected on the frequency tuned to.

Parameters

Table 912

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	The validity status of both the notified signal presence and the notified actual frequency.
`actFreq`	The actual frequency tuned to.
`presence`	A boolean indicating whether the presence of a signal (something other than noise) has been detected (`True`) or not (`False`).

Return Values
None.

Precondition
`True`

Action
— Any callback compliant action

Postcondition
`True`

8.1.15.5.2.5 OnAfcFreqChanged

Signature
```
[async] Void OnAfcFreqChanged (
    [in] UInt32 cookie,
    [in] uhTuning_ValStat_t actFreqStatus,
    [in] UInt32 actFreq,
    [in] uhTuning_AfcValue_t afcValue );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the actual frequency has changed due to AFC.

Parameters

Table 913

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`actFreqStatus`	The validity status of the notified actual frequency.
`actFreq`	Input parameter indicating the new actual frequency.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
afcValue	Input parameter indicating the current AFC value.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

Postcondition
True

8.1.15.5.2.6 OnAfcLimitReached

Signature
```
[async] Void OnLimitReached (
    [in] UInt32 cookie,
    [in] uhTuning_ValStat_t actFreqStatus,
    [in] UInt32 actFreq,
    [in] uhTuning_AfcValue_t afcValue );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the frequency has reached one of the limits of the AFC window.

Parameters

Table 914

Name	Description
cookie	The value that was passed by the client at subscription time.
actFreqStatus	The validity status of the notified actual frequency.
actFreq	Input parameter indicating the actual frequency when the limit was reached.
afcValue	Input parameter indicating the AFC value when the limit was reached.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

8.1.15.5.3 uhlTuningChan

Qualifiers
None

Description
This interface provides control with respect to channel based tuning and channel table insertion.

Interface ID
`uuid( 31C0D868-CDF3-4a25-AB70-8A9162FD56F5 )`

8.1.15.5.3.1 GetMaxTableSize

Signature
```
uhErrorCode_t GetMaxTableSize ( [out] pUInt32 pFieldFrequency );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns the maximum channel table size (in table entries).

Parameters

Table 915

Name	Description
`pSize`	The maximum channel table size (in table entries) allowed.

Return Values
Standard

Precondition
`True`

Action
`None`

Postcondition

`*pFieldFrequency == Aavd.maxChannelTableSize`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.15.5.3.2 SetTable

Signature

```
uhErrorCode_t SetTable (
    [in] UInt32 id,
    [in] uhTuning_ChanTableType_t type,
    [in] UInt32 size,
    [in, size_is(size)] puhTuning_ChanTableData_t pData );
```

Qualifiers

— synchronous

Description

This function sets the channel table that will be used for channel selection and searching.

Parameters

Table 916

Name	Description
`id`	Input parameter indicating the identification of the channel table. The `GetTable` function will return this identification so that a client can check which table is set.
`type`	The type of channel table based on input type.
`size`	Input parameter indicating the size (in table entries) of the channel table.
`pData`	Input parameter indicating the channel table data.

Return Values

Standard

Precondition

```
size <= tuner.maxChannelTableSize
```

Action

— Modify `tuner.channelTableId`
— `Modify tuner.channelTableType`
— Modify `tuner.channelTableSize`
— Modify `tuner.channelTable`

Postcondition

— `tuner.channelTableId == id`
— `tuner.channelTableType == type`
— `tuner.channelTableSize == size`
— For each `i` in `{0..size-1}`
  — `tuner.channelTable[i] == pData[i]`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.15.5.3.3 GetTable

Signature

```
uhErrorCode_t GetTable (
    [out] pUInt32 pId,
    [out] puhTuning_ChanTableType_t pType );
```

Qualifiers

— synchronous

— thread-safe

Description

This function returns information about the channel table currently used for channel selection and searching.

Parameters

Table 917

Name	Description
`pId`	Output parameter indicating the ID of the channel table as set by the client.
`pType`	Output parameter indicating the type of the channel table as set by the client.

Return Values

Standard

Precondition

Aavd.`channelTableSize != 0`

Action

None

Postcondition

— `*pId == tuner.channelTableId`

— `*pType == tuner.channelTableType`

8.1.15.5.3.4 GetOffsetRange

Signature

```
uhErrorCode_t GetOffsetRange (
    [out] pInt32 pMinOffset,
    [out] pInt32 pMaxOffset );
```

Qualifiers

— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function returns the boundaries for the channel offset in Hz. This offset can be used to adjust the frequency tuned to when a channel is set.

Parameters

Table 918

Name	Description
`pMinOffset`	Output parameter returning the minimum channel offset in Hz.
`pMaxOffset`	Output parameter returning the maximum channel offset in Hz.

Return Values
Standard

Precondition
`True`

Action
None

Postcondition
— `*pMinOffset == Aavd.minChannelOffset`
— `*pMaxOffset == tuner.maxChannelOffset`

8.1.15.5.3.5 SetChannel

Signature
```
uhErrorCode_t SetChannel (
    [in] UInt32 channel,
    [in] Int32 offset,
    [in] Bool fineTune );
```

Qualifiers
— asynchronous

Description
This function sets a channel based on the current channel table. A client can influence the frequency tuned to by specifying a frequency offset or by requesting a fine-tune action. If another search was already in progress, that search will be stopped by calling this function.

The following combinations are allowed for the offset and fine-tune parameters:

Table 919

offset	fine-tune	effect
[min .. max]	False	The indicated offset is applied, no fine-tune is done.
0	False	No offset is applied, no fine-tune is done.
0 or [min .. max]	True	Offset is ignored, only a fine-tune is done.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 920

Name	Description
`channel`	Input parameter indicating the channel to set.
`offset`	Input parameter indicating the frequency offset in Hz to apply.
`fineTune`	Input parameter indicating if a fine-tune action should be done (`True`) or not (`False`).

Return Values
Standard

Precondition
- `tuner.channelTableSize != 0`
- `channel in Aavd.channelTable`
- `tuner.minChannelOffset ≤ offset ≤ tuner.maxChannelOffset`

Action
- if `(tuner.tuning == True )`
  - Notify `OnAborted (tuner.actFreq )`
- if `(tuner.searching == True )`
  - Modify `Aavd.stopSearch`
- Modify `tuner.setChannel`
- Modify `tuner.actChannel`
- Modify `tuner.channelOffset`
- Modify `tuner.setChannelFineTune`
- `Modify tuner.fineTune`
- `Modify tuner.tuning`

Postcondition
- `tuner.stopSearch == True`
- `tuner.setChannel == channel`
- `tuner.actChannelStatus != uhTuning_ValStatKnown`
- `tuner.channelOffset == offset`
- `tuner.setChannelFineTune == fineTune`
- `tuner.fineTune == fineTune`
- `tuner.tuning == True`

Asynchronous Action
- Execute tune action
- if `(tuner.fineTune )`
  - `Execute fine tune action`
- Modify `tuner.actFreq`
- `Modify tuner.actChannel`
- `Modify tuner.fineTune`
- `Modify tuner.tuning`
- `Notify  OnTuned(tuner.actFreq )`

Asynchronous Postcondition
- if (`tuner.fineTune`)
  - `tuner.actFreq == round( finetune( carrier frequency of channel ) )`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— else
  — tuner.actFreq == round(tune(carrier frequency of channel + tuner.channelOffset))
— if (Aavd.actFreq is not part of a channel defined within the current channel table)
  — tuner.actChannelStatus != uhTuning_ValStatKnown
— else
  — tuner.actChannelStatus != uhTuning_ValStatUnknown
  — tuner.actChannel == channel ID for tuner.actFreq based on current channel table
— tuner.fineTune == False
— tuner.tuning == False

Remarks
— The actual frequency (tuner.actFreq) is rounded to the nearest frequency that is supported by the implementation.

8.1.15.5.3.6 GetChannel

Signature
```
uhErrorCode_t GetChannel (
    [out] pUInt32 pChannel,
    [out] pInt32  pOffset,
    [out] pBool   pFineTune );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the channel parameters as they where set by the last SetChannel function call.

Parameters

Table 921

Name	Description
pChannel	Output parameter returning the channel number that was set.
pOffset	Output parameter returning the frequency offset in Hz that was set.
pFineTune	Output parameter returning the fine tune parameter that was as set.

Return Values
Standard

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— *pChannel == tuner.setChannel
— *pOffset == tuner.channelOffset
— *pFineTune == tuner.fineTune

8.1.15.5.3.7 GetActualChannel

Signature
```
uhErrorCode_t GetActualChannel (
    [out] puhTuning_ValStat_t pChannelStatus,
    [out] pUInt32 pChannel );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the actual channel based on the actual frequency tuned to.

Parameters

Table 922

Name	Description
pChannelStatus	Output parameter returning the validity status of the actual channel. If a tune action is still busy (the frequency has not settled yet), in that case the value of pChannelStatus will be set to uhTuning_ValStatUnknown. If the tune action has finished and the actual frequency tuned to is not part of a channel defined within the current channel table, pChannelStatus will indicate uhTuning_ValStatUnknown.
pChannel	Output parameter returning the actual channel.

Return Values
Standard

Precondition
True

Action
None

Postcondition
— if (tuner.tuning == True) OR (tuner.searching == True)
  — tuner.actChannelStatus != uhTuning_ValStatKnown
— else if (tuner.actFreq not in channel of channel table)
  — tuner.actChannelStatus != uhTuning_ValStatKnown
— else
  — tuner.actChannelStatus != uhTuning_ValStatUnknown
— *pChannelStatus == tuner.actChannelStatus

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— *pChannel == tuner.actChannel

8.1.15.5.4 uhITuningSearch

Qualifiers
None.

Description
This interface provides control with respect to searching.

Interface ID
uuid(505C46FA-EC96-422a-8907-BF5C102E7D18)

8.1.15.5.4.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhITuningSearchNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhTuning_SearchNotifications_t notifs );
```

Qualifiers
— subscribe-function

8.1.15.5.4.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhITuningSearchNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhTuning_SearchNotifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.1.15.5.4.3 GetSuppModes

Signature
```
uhErrorCode_t GetSuppModes (
    [out] puhTuning_SearchMode_t pModes );
```

Qualifiers
— synchronous

— thread-safe

Description
This function returns the supported search modes.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 923

Name	Description
pModes	Output parameter indicating the supported search modes.

Return Values
Standard

Precondition
True

Action
None

Postcondition
*pModes == Aavd.suppSearchModes

8.1.15.5.4.4 Start

Signature
```
uhErrorCode_t Start (
    [in] uhTuning_SearchMode_t mode,
    [in] UInt32 endPos );
```

Qualifiers
— asynchronous

Description
This function starts a search from the current position using the specified search modes. The search ends when a signal is found, the end position has been reached or the search has been stopped. In all cases, a notification will indicate the termination of the search. When a search reaches the end of the search range (when searching upwards) the position will wrap to the beginning of the range and vice versa (when searching downwards). The following table gives an overview of the constraints with respect to the search mode flags that can be set in order to be valid.

Table 924

Mode flags	Constraint
Modulation type	If negative, positive modulation and modulation none is supported, they are mutually exclusive (the mode being set may not specify multiple at the same time).
Search type	If frequency and channel based searching is supported, they are mutually exclusive (the mode being set may not specify both at the same time).
Search direction	If downward and upward searching is supported, they are mutually exclusive (the mode being set may not specify both at the same time).
Current position included	If current position included and current position excluded searching is supported, they are mutually exclusive (the mode being set may not specify both at the same time).
FineTuning	If fine-tune included and fine-tune excluded searching is supported, they are mutually exclusive (the mode being set may not specify both at the same time).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 925

Name	Description
mode	Input parameter indicating the search mode. The mode that is set has to be supported and has to be valid.
endPos	Input parameter indicating the position at which the search has to stop. The unit of the position depends on the type of search. For a frequency based search the unit is Hz and for a channel based search the unit is the channel number.

Return Values
Standard

Precondition
— mode in tuner.suppSearchModes and valid (see constraints on mode flag parameters)
— if (mode indicates a frequency based search)
  — tuner.minAmp ≤ endPos ≤ tuner.maxFreq
— else
  — tuner.channelTableSize != 0
  — endPos in the range of channel numbers of the current channel table

Action
— if (tuner.stopSearch == True)
  — Modify tuner.stopSearch
  — Modify tuner.searching
  — Notify OnStopped(tuner.actFreq)
— else if (tuner.searching == True)
  — Notify OnStopped(tuner.actFreq)
— else
  — Modify tuner.searching
  — Modify tuner.actChannel

Postcondition
— if (tuner.stopSearch == True)
  — tuner.stopSearch == False
  — tuner.searching == False
  — Do not execute the asynchronous part
— else
  — tuner.searching == True
  — tuner.actChannelStatus != uhTuning_ValStatKnown

Asynchronous Action
— Modify tuner.searching
— Modify tuner.actFreq
— Modify tuner.actChannel
— Modify tuner.signalPresence
— if (tuner.stopSearch == True)

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

    - Modify tuner.stopSearch
    - Notify OnStopped(tuner.actFreq)
- if (signal found)
    - Notify OnSignalDetected(Aavd.actFreq)
- if (tuner.actFreq == frequency of endPos)
    - Notify OnEndPosReached(tuner.actFreq)

Asynchronous Postcondition
- tuner.searching == False
- tuner.actFreq == current frequency
- if (Aavd.stopSearch == True)
    - tuner.stopSearch == False
- if (tuner.actFreq part of channel defined within current channel table)
    - tuner.actChannel == current channel based on channel table
- else
    - tuner.actChannelStatus != uhTuning_ValStatKnown
- if (signal found)
    - tuner.signalPresence == True
- else
    - tuner.signalPresence == False

8.1.15.5.4.5 Stop

Signature
```
uhErrorCode_t Stop ();
```

Qualifiers
- synchronous

Description
This function can be used to stop a search. If a search was in progress and has been stopped the client will be notified if subscribed to OnStopped.

Parameters
None.

Return Values
Standard

Precondition
True

Action
- if (tuner.searching == True)
    - Modify tuner.stopSearch

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`tuner.stopSearch == True`

8.1.15.5.5 uhITuningSearchNtf

Qualifiers
— callback

Description
This interface can be provided by a client that wants to receive notifications with respect to searching.

Interface ID
`uuid( F1EC46F5-FB4F-4f8c-B2AD-A443029B95B9 )`

8.1.15.5.5.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhTuning_SearchNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.15.5.5.2 OnSignalDetected

Signature
```
[async] Void OnSignalDetected (
    [in] UInt32 Cookie,
    [in] uhTuning_ValStat_t actFreqStatus,
    [in] UInt32 actFreq );
```

Qualifiers
None.

Description
This function notifies subscribed clients that a signal has been detected and the search has been stopped.

Parameters

Table 926

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`actFreqStatus`	The validity status of the notified actual frequency.
`actFreq`	Input parameter indicating the actual frequency the signal was detected on.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.15.5.5.3 OnStopped

Signature
```
[async] Void OnStopped (
    [in] UInt32 cookie,
    [in] UInt32 actFreq );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the search has been stopped.

Parameters

Table 927

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`actFreq`	Input parameter indicating the actual frequency after the searching has been stopped.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.15.5.5.4 OnEndPosReached

Signature

```
[async] Void OnEndPosReached (
    [in] UInt32 cookie,
    [in] UInt32 freq );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the search has stopped because the end position that was set using the `Start` function has been reached.

Parameters

Table 928

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`freq`	Input parameter indicating the frequency of the end position.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.1.15.5.5.5 OnFreqChanged

Signature

```
[async] Void OnFreqChanged (
    [in] UInt32 cookie,
    [in] UInt32 freq );
```

Qualifiers
None

Description
This function notifies subscribed clients that the frequency has changed during searching. This function is intended for progress indication purposes.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 929

Name	Description
cookie	The value that was passed by the client at subscription time.
freq	Input parameter indicating the new frequency in Hz.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action.

Postcondition
True

8.1.15.5.5.6 OnChannelChanged

Signature

```
[async] Void OnChannelChanged (
    [in] UInt32 cookie,
    [in] UInt32 channel );
```

Qualifiers
None.

Description
This function notifies subscribed clients that the channel has changed during searching. This function is intended for progress indication purposes.

Parameters

Table 930

Name	Description
cookie	The value that was passed by the client at subscription time.
channel	Input parameter indicating the new channel number.

Return Values
None.

Precondition
True

Action
— Any call-back compliant action

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

8.1.16 UhlUrlSrc

8.1.16.1 Concepts

8.1.16.1.1 Introduction

The function of a URL Source is to stream data identified by a URL into the platform. A URL has the following format: `<scheme>://<scheme specific format>`[22]. The scheme, also called protocol, is used for accessing the data. Examples are: FILE, HTTP, and FTP. For the HTTP scheme, a URL will look like: `http://<hostport>[/<path>][?<search>]`[23].

Based on the protocol, capabilities of the server, and content type of the data special operations can be applied to the data like pause, fast forward, go to position, and so on. These special operations are often referred to as trick modes. When a new URL is requested, the available trick modes may change. Clients have to take this into account. Note that linear playback of the data is always supported.

Accessing a server that provides a URL can be slow, especially when it points to a remote server. Therefore, gaining access is done in two steps. First, the URL is parsed to see if the requested scheme is available and the format of the URL is correct. This can be done locally and fast. Secondly, the server and the actual data will be accessed which might require remote access and can be slow.

When a URL identifies data on a remote server, the connection to the server may be lost. This will be signaled to the client. A number of retries will be performed before a definite connection lost notification is given. Furthermore, delays in the transmission can occur. These will not be signaled to the client but gracefully handled by the platform.

Data streaming from a server is subject to jitter. Therefore, an input buffer is modeled as part of URL Source. This buffer is used to compensate for this jitter, that is, to ensure a continuous playback. When the URL Source is put in play state, data streaming to the output only starts once the amount of data in the buffer has reached a nominal level. Then, the fullness level of the buffer fluctuates around (below and above) the nominal level. The detailed buffering model during trick modes is scheme-specific and not described further in this document.

The ratio between the nominal level and the total buffer size is URL scheme-specific. The total buffer size itself depends on the maximum possible jitter in the system and on the bit rate of the selected URL. The fullness level of the buffer is observable to the client (in % of nominal level).

[22] Terms in pointy brackets have to be substituted by an actual string and the brackets have to removed.

[23] Terms in square brackets are optional and the brackets have to be removed.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.2 Types & Constants

8.1.16.2.1 Public Types & Constants

8.1.16.2.1.1 Error Codes

Signature

```
const uhErrorCode_t UH_ERR_URLSRC_INVALID_URL      = 0x00000801;
const uhErrorCode_t UH_ERR_URLSRC_SCHEME_NOT_SUPP  = 0x00000802;
const uhErrorCode_t UH_ERR_URLSRC_ABORTED          = 0x00000803;
const uhErrorCode_t UH_ERR_URLSRC_NOT_ABORTED      = 0x00000804;
const uhErrorCode_t UH_ERR_URLSRC_END_OF_STREAM    = 0x00000805;
const uhErrorCode_t UH_ERR_URLSRC_CONNECTION_LOST  = 0x00000806;
```

Qualifiers

— error-codes

Description

The non-standard error codes that can be returned by functions of this logical component.

Constants

Table 931

Name	Description
UH_ERR_URLSRC_INVALID_URL	This error is returned when a URL cannot be parsed.
UH_ERR_URLSRC_SCHEME_NOT_SUPP	This error is returned when a scheme indicated in the URL is not supported.
UH_ERR_URLSRC_ABORTED	A call to abort an action indeed aborted the action.
UH_ERR_URLSRC_NOT_ABORTED	A call to abort an action did not abort the action.
UH_ERR_URLSRC_END_OF_STREAM	A call to a single step function has reached the end of stream and thus the single step is not performed.
UH_ERR_URLSRC_CONNECTION_LOST	A call to a single step function is interrupted because of a lost connection.

8.1.16.2.1.2 uhUrlSrc_Notification_t

Signature

```
typedef enum _uhUrlSrc_Notification_t {
    uhUrlSrc_SubscriptionChanged = 0x00000001,
    uhUrlSrc_Accessible          = 0x00000002,
    uhUrlSrc_NotAccessible       = 0x00000004,
    uhUrlSrc_ConnectionRetry     = 0x00000008,
    uhUrlSrc_EndOfRendering      = 0x00000010,
    uhUrlSrc_EndOfInput          = 0x00000020,
    uhUrlSrc_DataConsumed        = 0x00000040,
    uhUrlSrc_PlayTimeChanged     = 0x00000080,
    uhUrlSrc_ActionAborted       = 0x00000100,
    uhUrlSrc_TrickControlLost    = 0x00000200
} uhUrlSrc_Notification_t, *puhUrlSrc_Notification_t;
```

Qualifiers

— enum-element

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIUrlSrcNtf`.

Values

Table 932

Name	Description
`uhUrlSrc_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhUrlSrc_Accessible`	Represents `OnAccessible`.
`uhUrlSrc_NotAccessible`	Represents `OnNotAccessible`.
`uhUrlSrc_ConnectionRetry`	Represents `OnConnectionRetry`.
`uhUrlSrc_EndOfRendering`	Represents `OnEndOfRendering`.
`uhUrlSrc_EndOfInput`	Represents `OnEndOfInput`.
`uhUrlSrc_DataConsumed`	Represents `OnDataConsumed`.
`uhUrlSrc_PlayTimeChanged`	Represents `OnPlayTimeChanged`.
`uhUrlSrc_ActionAborted`	Represents `OnActionAborted`.
`uhUrlSrc_TrickControlLost`	Represents `OnTrickControlLost`.

8.1.16.2.1.3 uhUrlSrc_Notifications_t

Signature
```
typedef Int32 uhUrlSrc_Notifications_t, *puhUrlSrc_Notifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type `uhUrlSrc_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints
None.

8.1.16.2.1.4 uhUrlSrc_AllNotifications

Signature
```
const uhUrlSrc_Notifications_t uhUrlSrc_AllNotifications = 0x000003FF;
```

Qualifiers
None.

Description
Defines the set of all values of type `uhUrlSrc_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.2.1.5 uhUrlSrc_Scheme_t

Signature

```
typedef enum _uhUrlSrc_Scheme_t {
    uhUrlSrc_SchemeFtp      = 0x00000001,
    uhUrlSrc_SchemeHttp     = 0x00000002,
    uhUrlSrc_SchemeHttps    = 0x00000004,
    uhUrlSrc_SchemeGopher   = 0x00000008,
    uhUrlSrc_SchemeNews     = 0x00000010,
    uhUrlSrc_SchemeNntp     = 0x00000020,
    uhUrlSrc_SchemeFile     = 0x00000040,
    uhUrlSrc_SchemeProspero = 0x00000080,
    uhUrlSrc_SchemeRtsp     = 0x00000100,
    uhUrlSrc_SchemeRtspu    = 0x00000200
} uhUrlSrc_Scheme_t, *puhUrlSrc_Scheme_t;
```

Qualifiers

— enum-element

Description

Values that describe the possible scheme supported by this URL Source component. The enumeration contains only the schemes for which this URL Source definition is an appropriate abstraction.

Values

Table 933

Name	Description
uhUrlSrc_SchemeFtp	File transfer protocol.
uhUrlSrc_SchemeHttp	Hypertext Transfer Protocol.
uhUrlSrc_SchemeHttps	Secure HTTP.
uhUrlSrc_SchemeGopher	The Gopher protocol.
uhUrlSrc_SchemeNews	USENET news.
uhUrlSrc_SchemeNntp	USENET news using NNTP access
uhUrlSrc_SchemeFile	Host-specific file names.
uhUrlSrc_SchemeProspero	Prospero Directory Service.
uhUrlSrc_SchemeRtsp	Real-time Streaming Protocol using TCP
uhUrlSrc_SchemeRtspu	Real-time Streaming Protocol using UDP

8.1.16.2.1.6 uhUrlSrc_Schemes_t

Signature

```
typedef Int32 uhUrlSrc_Schemes_t, *puhUrlSrc_Schemes_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of schemes.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.2.1.7 uhUrlSrc_NotAccessibleCause_t

Signature

```
typedef enum _uhUrlSrc_NotAccessibleCause_t {
    uhUrlSrc_NacNotFound             = 0,
    uhUrlSrc_NacTimeOut              = 1,
    uhUrlSrc_NacConnectionLost       = 2,
    uhUrlSrc_NacConnectionTerminated = 3
} uhUrlSrc_NotAccessibleCause_t, *puhUrlSrc_NotAccessibleCause_t;
```

Qualifiers
None.

Description
Values that describe possible causes of a server not being accessible.

Values

Table 934

Name	Description
uhUrlSrc_NacNotFound	The URL is not found.
uhUrlSrc_NacTimeOut	The URL is not found in the specified time.
uhUrlSrc_NacConnectionLost	The connection is lost (while it was previously established).
uhUrlSrc_NacConnectionTerminated	The connection was terminated by the client.

8.1.16.2.1.8 uhUrlSrc_Rounding_t

Signature

```
typedef enum _uhUrlSrc_Rounding_t {
    uhUrlSrc_RoundUp      = 0,
    uhUrlSrc_RoundDown    = 1,
    uhUrlSrc_RoundNearest = 2
} uhUrlSrc_Rounding_t, *puhUrlSrc_Rounding_t;
```

Qualifiers
None.

Description
Values that describe the rounding technique used when setting a position in a stream.

Values

Table 935

Name	Description
uhUrlSrc_RoundUp	Round up.
uhUrlSrc_RoundDown	Round down.
uhUrlSrc_RoundNearest	Round nearest.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.2.1.9 uhUrlSrc_StepMode_t

Signature

```
typedef enum _uhUrlSrc_StepMode_t {
    uhUrlSrc_StepForward  = 0x00000001,
    uhUrlSrc_StepBackward = 0x00000002
} uhUrlSrc_StepMode_t, *puhUrlSrc_StepMode_t;
```

Qualifiers

— enum-element

Description

Values of this type represent step modes.

Values

Table 936

Name	Description
uhUrlSrc_StepForward	Step forward.
uhUrlSrc_StepBackward	Step backward.

8.1.16.2.1.10 uhUrlSrc_StepModes_t

Signature

```
typedef Int32 uhUrlSrc_StepModes_t, *puhUrlSrc_StepModes_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of step modes.

8.1.16.2.2 Model Types & Constants

8.1.16.2.2.1 AccessState_t

Signature

```
typedef enum _AccessState_t {
    AsNoAccess   = 0,
    AsAccessing  = 1,
    AsAccessible = 2,
    AsRetrying   = 3
} AccessState_t, *pAccessState_t;
```

Qualifiers

None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values describing the states of the `UrlSrc` role regarding access.

Values

Table 937

Name	Description
`AsNoAccess`	The URL is not accessible.
`AsAccessing`	The component is trying to obtain access to a URL.
`AsAccessible`	The URL is accessible.
`AsRetrying`	The `UrlSrc` role is trying to re-access the server of the URL.

8.1.16.2.2.2 PlayState_t

Signature
```
typedef enum _PlayState_t {
    PsStopped = 0,
    PsPlaying = 1,
    PsPaused  = 2
} PlayState_t, *p PlayState_t;
```

Qualifiers
None.

Description
Values describing the states of the `UrlSrc` role regarding playing.

Values

Table 938

Name	Description
`PsStopped`	The `UrlSrc` role is not playing the data.
`PsPlaying`	The `UrlSrc` role is playing the data when there is input provided.
`PsPaused`	The `UrlSrc` role is paused.

8.1.16.3 Logical Component

8.1.16.3.1 Interface-Role Model

Figure 97 depicts the interface-role model of the URL Source logical component (grey box). It shows the interfaces and the roles involved with this component.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

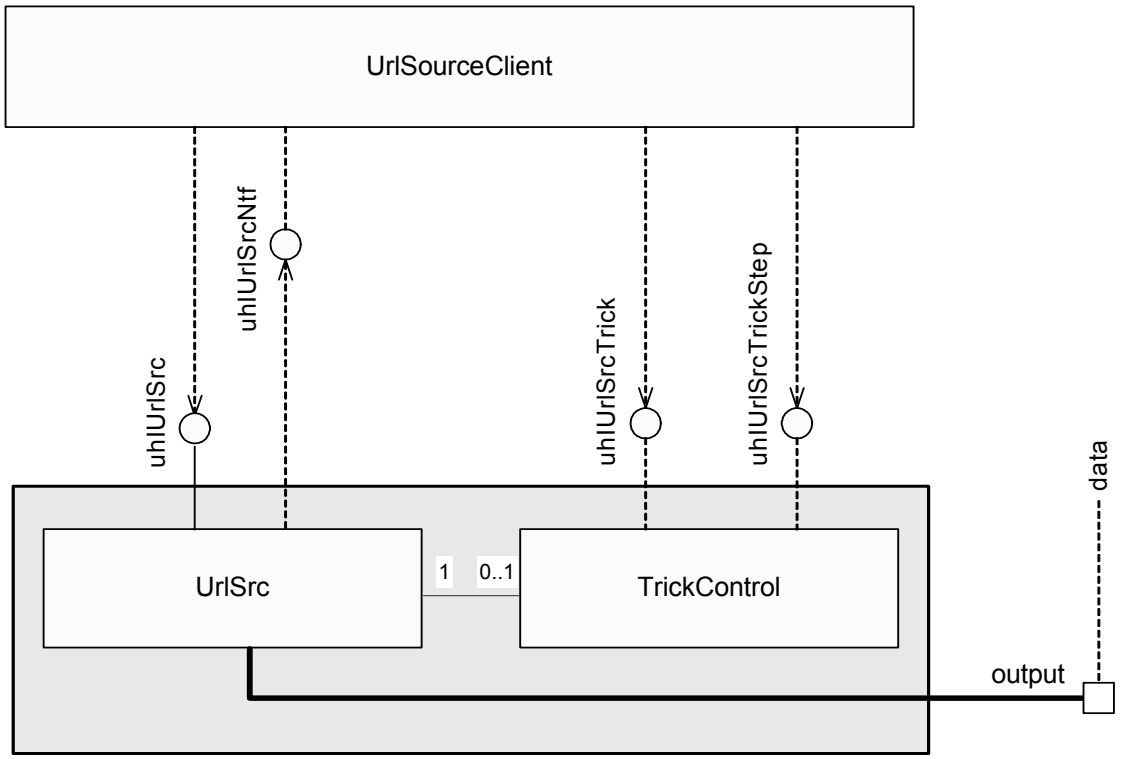


Figure 97 — Interface Role Model

A URL Source logical component contains the `UrlSrc` role that provides a mandatory control interface and can provide the client with notifications. It also contains the dynamic `ImageDecodingClient` role that only provides optional (dotted line) control interfaces. The `ImageDecodingClient` role is dynamic because it can become unavailable at run-time. When the role becomes unavailable, the interfaces it provides will also become unavailable and should be released by a client. The `ImageDecodingClient` role will be unavailable when the client sets a new URL, with the exception that when the new host serving the content has the same capabilities and the content type does not change. When the `ImageDecodingClient` role becomes unavailable, `OnTrickControlLost` is called.

A client (represented by the `UrlSourceClient` role) can use the provided control interfaces. The client can optionally provide a notification interface in order to receive notifications of events generated by the URL Source logical component.

8.1.16.3.2 Diversity

8.1.16.3.2.1 Provided Interfaces

Table 939

Role	Interface	Presence
`UrlSrc`	`uhIUrlSrc`	mandatory
`UrlSourceClient`	`uhIUrlSrcNtf`	optional
`ImageDecodingClient`	`uhIUrlSrcTrick`	optional
`ImageDecodingClient`	`uhIUrlSrcTrickStep`	optional

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.3.2.2 Configurable Items

Table 940

Role	Attribute
UrlSrc	callbackRate
UrlSrc	nrOfRetries
UrlSrc	supportedSchemes
ImageDecodingClient	availStepModes
ImageDecodingClient	reverseSupp
ImageDecodingClient	minSpeedNum
ImageDecodingClient	minSpeedDenom
ImageDecodingClient	maxSpeedNum
ImageDecodingClient	maxSpeedDenom

8.1.16.3.2.3 Constraints

None.

8.1.16.3.3 Instantiation

8.1.16.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 941

Type	Object	Multiplicity
UrlSrc	urlSource	1
ImageDecodingClient	trickControl	0..1

8.1.16.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

- urlSource.accessState == AsNoAccess
- urlSource.currentUrl == empty string
- urlSource.playState == PsStopped
- urlSource.pTrickControl == Null
- urlSource.totalSize == 0
- urlSource.totalDuration == 0

8.1.16.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier “thread-safe”.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.4 Roles

8.1.16.4.1 UrlSrc

Signature

```
role UrlSrc{
    AccessState_t              accessState;
    PlayState_t                playState;
    String                     currentUrl;
    const UInt32               callbackRate;
    const UInt32               nrOfRetries;
    TrickControl*              pTrickControl;
    const uhUrlSrc_Schemes_t   supportedSchemes;
    UInt32                     totalSize;
    UInt32                     totalDuration;
    UInt32                     nominalBufferLevel;
    UInt32                     currentBufferLevel;
}
```

Qualifiers

— root

Description

The UrlSrc role provides functionality to access a URL and stream its data into the platform. By default, linear playback is supported. It can provide notifications to its clients.

Independent Attributes

Table 942

Name	Description
accessState	The current access state.
playState	The current play state.
currentUrl	The URL identifying the current data to play.
callbackRate	Indicates the maximum number of times per second the notification functions OnDataConsumed and OnPlayTimeChanged will be called.
nrOfRetries	Indicates the number of retries the UrlSrc role will try before a notification OnNotAccessible is given. This is only done when the connection was previously established and has been lost due to external events.
pTrickControl	Points to the optionally available ImageDecodingClient object (Null if not available).
supportedSchemes	A configurable item indicating the URL schemes (see subclause 8.1.16.1) are supported by this component.
totalSize	The total size of the data represented by the current URL
totalDuration	The total duration of the data represented by the current URL
nominalBufferLevel	The nominal buffer level for the current URL.
currentBufferLevel	The current buffer fullness level.

Invariants

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Instantiation
See subclause 8.1.16.3.3

Streaming Behavior
The `UrlSrc` has no streaming input, because it is a source. It streams data (e.g. audio, video) from a physical source indicated by a URL. When the data is not streamed fast enough the platform will implement graceful degradation.

When the input of a stream indicates that all data is read, a notification `OnEndOfInput` is signaled. In this case, it is possible that the stream is still playing, but the component is then ready for a new call to `OpenUrl`.

When all data of a URL has been rendered then `OnEndOfRendering` is notified. In this case there is no autonomous state change (for instance `playState`) in the `UrlSrc`.

The preconditions on the `playState` in the `Stop`, `Start` (and `Pause`, `Resume` for role `ImageDecodingClient`) are independent of autonomous behavior. The `playState` is thus fully deducible from the calls to the mentioned functions and therefore not made observable.

When the `UrlSrc` is stopped (`Stop`), the position in the stream is reset to the start of that stream if the `UrlSrc` can negotiate about the position with the server. In this case, a (re)start (`Start`) will be from that start position. Otherwise the (re)start picks up from the data coming in.

Active Behavior
If the connection to a server is lost, a number (`nrOfRetries`) of retries is performed. In this case, the `accessState` is set to `AsRetrying`. Each retry is notified. If the retries are not aborted (see `AbortAction`), (1) after `nrOfRetries` retries an `OnNotAccessible` is sent or (2) when access is regained an `OnAccessible` is notified.

8.1.16.4.2 TrickControl

Signature

```
role TrickControl{
    const uhUrlSrc_StepModes_t availStepModes;
    const Bool                 reverseSupp;
    const UInt8                minSpeedNum;
    const UInt8                minSpeedDenom;
    const UInt8                maxSpeedNum;
    const UInt8                maxSpeedDenom;
    const uhUrlSrc_StepModes_t actualAvailStepModes;
    const Bool                 actualReverseSupp;
    const UInt8                actualMinSpeedNum;
    const UInt8                actualMinSpeedDenom;
    const UInt8                actualMaxSpeedNum;
    const UInt8                actualMaxSpeedDenom;
    Int8                       requestedSpeedNum;
    UInt8                      requestedSpeedDenom;
    Int8                       actualSpeedNum;
    UInt8                      actualSpeedDenom;
    UInt32                     currentPosition;
}
```

Qualifiers
— actor

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description

The TrickControl role is dynamic. Depending on the content type and server capabilities of the data in the requested URL, it provides functionality to apply trick modes on the data that is streamed. This is only possible when the data rate is controlled by the UrlSrc component. The constant attributes of a TrickControl instance mirror the trick modes as allowed by the content type and server capabilities.

Independent Attributes

Table 943

Name	Description
availStepModes	The available step modes support by the platform.
reverseSupp	When True indicates that the speed range (as indicated by the other parameters) can also be used for reverse speed for this platfrom.
minSpeedNum	The numerator of the lower limit for the play back speed as support by the platform.
minSpeedDenom	The denominator of the lower limit for the playback speed as support by the platform.
maxSpeedNum	The numerator of the upper limit for the playback speed as support by the platform.
maxSpeedDenom	The denominator of the upper limit for the playback speed as support by the platform.
requestedSpeedNum	The numerator part of the requested playback speed.
actualAvailStepModes	The actual available step modes taking the capabilities of the platform, the capabilities of the server, and the content type into account.
actualReverseSupp	When True indicates that the actual speed range (as indicated by the other parameters) can also be used for reverse speed taking the capabilities of the platform, the capabilities of the server, and the content type into account.
actualMinSpeedNum	The actual numerator of the lower limit for the play back speed taking the capabilities of the platform, the capabilities of the server, and the content type into account.
actualMinSpeedDenom	The actual denominator of the lower limit for the playback speed taking the capabilities of the platform, the capabilities of the server, and the content type into account.
actualMaxSpeedNum	The actual numerator of the upper limit for the playback speed taking the capabilities of the platform, the capabilities of the server, and the content type into account.
actualMaxSpeedDenom	The actual denominator of the upper limit for the playback speed taking the capabilities of the platform, the capabilities of the server, and the content type into account.
requestedSpeedDenom	The denominator part of the requested playback speed.
actualSpeedNum	The numerator part of the actual playback speed.
actualSpeedDenom	The denominator part of the actual playback speed.
currentPosition	The current position in the stream that is playing.

Invariants

- minSpeedDenom != 0
- maxSpeedNum != 0
- maxSpeedDenom != 0
- actualMinSpeedDenom != 0
- actualMaxSpeedNum != 0
- actualMaxSpeedDenom != 0
- requestedSpeedDenom != 0
- actualSpeedDenom != 0
- actualAvailStepModes is a subset of availStepModes

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— If reverse play is supported by the current server and reverse play is applicable for the current content type
  — `actualReverseSupp == reverseSupp`
— Else
  — `actualReverseSupp == False`
— `(minSpeedNum/minSpeedDenom) <= (actualMinSpeedNum/actualMinSpeedDenom) <= (actualMaxSpeedNum/actualMaxSpeedDenom) <= (maxSpeedNum/maxSpeedDenom)`
— If `reverseSupp == True`
  — `(actualMinSpeedNum/actualMinSpeedDenom) <= (abs(requestedSpeedNum)/requestedSpeedDenom) <= (actualMaxSpeedNum/actualMaxSpeedDenom)`
  — `(actualMinSpeedNum/actualMinSpeedDenom) <= (abs(actualSpeedNum)/actualSpeedDenom) <= (actualMaxSpeedNum/actualMaxSpeedDenom)`
— Else
  — `(actualMinSpeedNum/actualMinSpeedDenom) <= (requestedSpeedNum/requestedSpeedDenom) <= (actualMaxSpeedNum/actualMaxSpeedDenom)`
  — `(actualMinSpeedNum/minSpeedDenom) <= (actualSpeedNum/actualSpeedDenom) <= (actualMaxSpeedNum/actualMaxSpeedDenom)`

Instantiation
An instance of this role is created after the server, as requested in the URL, has become available and the content type of the data is known. Based on the content type and server capabilities it is determined what trick modes can be provided via the `uhIUrlSrcTrick` or `uhIUrlSrcTrickStep` interface. When at least the trick modes `Pause`, `Resume`, and `SetPosition` are available, the `trickControl` object can be created. The object is created with a call to `GetTrickControl`.

The values of `actualAvailStepModes`, `actualReverseSupp`, `actualMinSpeedNum`, `actualMinSpeedDenom`, `actualMaxSpeedNum`, and `actualMaxSpeedDenom` are determined by the platform based on the capabilities of the platform, the capabilities of the server, and the content type.

Initial State

— `requestedSpeedNum == 1`
— `requestedSpeedDenom == 1`
— `actualSpeedNum == 1`
— `actualSpeedDenom == 1`
— `currentPosition` is determined from the position of the `UrlSrc` role.

Streaming Behavior
None.

Active Behavior
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.4.3 UrlSourceClient

Signature
```
role UrlSourceClient{}
```

Qualifiers
— actor

Description
A UrlSourceClient represents a user of the `uhIUrlSrc` `uhIUrlSrcTrick`, and `uhIUrlSrcTrickStep` interfaces. It can provide the `uhIUrlSrcNtf` interface when it wants to receive notifications. No constraints apply to the behavior of the `UrlSourceClient` other than those implied by Figure 97.

8.1.16.5 Interfaces

8.1.16.5.1 uhIUrlSrc

Qualifiers
— None

Description
This interface provides functionality to access a URL and provide linear playback. Typically, the functions in this interface should be used in a particular order which is: `Subscribe`, `OpenUrl`, wait for `OnAccessible`, and then `Start`.

Interface ID
```
uuid( 4CE3FDA6-25BF-4661-9E52-ACD03EFE1E61 )
```

8.1.16.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIUrlSrcNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhUrlSrc_Notifications_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIUrlSrcNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhUrlSrc_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.16.5.1.3 GetSuppSchemes

Signature

```
uhErrorCode_t GetSuppSchemes ( [out] puhUrlSrc_Schemes_t pSupportedSchemes );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns all supported schemes (see 8.1.16.2.1.5).

Parameters

Table 944

Name	Description
`pSupportedSchemes`	Output parameter returning the supported schemes.

Return Values

Standard

Precondition

`True`

Action

— None

Postcondition

— *`*pSupportedSchemes == urlSource.supportedSchemes.`*

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.5.1.4 OpenUrl

Signature

```
uhErrorCode_t OpenUrl (
    [in, string] const String url,
    [in] UInt32 timeOut );
```

Qualifiers

— asynchronous

— single-threaded

Description

Sets the URL that identifies the data. The URL should comply to [29]. The `OpenUrl` function will try to make the connection to the location. It will generate a notification `OnNotAccessible` or `OnAccessible` on failure or on success.

During the attempt to access the URL, the state of the `UrlSrc` is `AsAccessing`. When the `UrlSrc` is in this state no other `OpenUrl` may be called. This state is not explicitly observable but only implicitly by deducing it from the notifications.

Parameters

Table 945

Name	Description
`url`	The URL that identifies the data that has to be played. The string is null-terminated.
`timeOut`	The number of microseconds (larger than 0) that building a connection may take before it is automatically cancelled. The actual time-out may depend on the underlying platform.

Return Values

Table 946

Name	Description
`UH_OK`	Indicates successful execution.
`UH_ERR_URLSRC_INVALID_URL`	The requested URL could not be parsed.
`UH_ERR_URLSRC_SCHEME_NOT_SUPP`	The scheme (see subclause 8.1.16.1.1) of the URL is not supported.

Precondition

— `urlSource.accessState != AsAccessing`

— *timeOut* `> 0`

Action

— If *url* is valid URL
  — Modify `urlSource.currentUrl`
  — Modify `urlSource.accessState`
  — `Flush input buffer`
  — Modify `urlSource.currentBufferLevel`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
- If *url* is valid URL && scheme of `url` supported
  - `urlSource.currentUrl` is a copy of *url*
  - `urlSource.accessState == AsAccessing`
  - `urlSource.currentBufferLevel == 0`

Asynchronous Action
- Modify `urlSource.accessState`
- If server and data in URL are accessible
  - If the server providing the URL does not have trick mode capabilities && `urlSource.pTrickControl != Null`
    - `urlSource.pTrickControl = Null`
    - Notify `OnTrickControlLost(*)`
  - Compute new `urlSource.nominalBufferLevel`
  - Notify `OnAccessible(*)`
- Else
  - If access was aborted due to a timeout
    - Notify `OnNotAccessible(*, uhUrlSrc_NacTimeOut)`
  - Else
    - Notify `OnNotAccessible(*, uhUrlSrc_NacNotFound)`

Asynchronous Postcondition
- If server and data in URL are accessible within time timeOut
  - `urlSource.accessState == AsAccessible`
  - `new urlSource.nominalBufferLevel set`
- Else
  - `urlSource.accessState == AsNoAccess`

Remarks
- When the component was in the playing state, it immediately starts playing the current URL.

- Note that when the `pTrickControl` is set to Null, this does not mean that the `pTrickControl` object is destroyed. Only when all interface pointers to that object are released, the object is destroyed. This is the obligation of the client.

8.1.16.5.1.5 GetUrl

Signature
```
uhErrorCode_t GetUrl (
    [in] UInt32  size,
    [out] pUInt32   pActualSize,
    [out, string, size_is(size)] String url);
```

Qualifiers
- synchronous

- thread-safe

Description
Returns the URL that was set as a null-terminated string and the actual size of the URL.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 947

Name	Description
`size`	The maximum number of characters to be returned of the URL identified by url.
`pActualSize`	Output parameter returning the actual number of characters of the URL identified by *url*.
`url`	Output parameter returning the URL that was set as a null-terminated string.

Return Values
Standard

Precondition
`True`

Action
— None

Postcondition
— `*`*pActualSize* == actual size of the URL identified by *url*
— *url* is a copy of `urlSource.currentUrl`

8.1.16.5.1.6 Start

Signature
```
uhErrorCode_t Start ( );
```

Qualifiers
— synchronous

— single-threaded

Description
This function starts streaming the data whenever data becomes available. A consecutive call to start does not have any effect on the URL currently being streamed.

Return Values
Standard

Precondition
— True

Action
— Modify `urlSource.playState`

Postcondition
`urlSource.playState == PsPlaying`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks

— Data becomes available only with a call to `OpenUrl`.

— See also 8.1.16.4.1 for details on streaming behavior for starting a stopped stream.

8.1.16.5.1.7 Stop

Signature
```
uhErrorCode_t Stop ( );
```

Qualifiers
— synchronous

— single-threaded

Description
Stops playing the current URL (flushes the data that has not been decoded, and stops rendering as soon as possible).

Return Values
Standard

Precondition
— True

Action
— If `urlSource.playState == PsPlaying || urlSource.playState == PsPaused`
  — Modify `urlSource.playState`
  — Notify `OnEndOfRendering(*)`
  — Flush input buffer
  — Modify `urlSource.currentBufferLevel`

Postcondition
— `urlSource.playState == PsStopped`
— `urlSource.currentBufferLevel == 0`

Remarks
— See also 8.1.16.4 for details on streaming behavior related to stopping a stream.

8.1.16.5.1.8 CloseUrl

Signature
```
uhErrorCode_t CloseUrl ( );
```

Qualifiers
— synchronous

— single-threaded

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Closes access to the server providing the URL.

Return Values
Standard

Precondition
`urlSource.accessState != AsAccessing`

Action
— If `urlSource.accessState == AsAccessible || urlSource.accessState == AsRetrying`
  — Modify `urlSource.accessState`
  — Notify `OnNotAccessible (*, uhUrlSrc_NacConnectionTerminated)`

Postcondition
`urlSource.accessState == AsNoAccess`

Remarks

— A UrlSrc is required to re-obtain access to the URL.

— Note that the play state is not changed.

8.1.16.5.1.9 AbortAction

Signature
`uhErrorCode_t AbortAction ( );`

Qualifiers
— synchronous

— single-threaded

Description
This function is intended to abort an `OpenUrl` or an automatic connection retry. Calling this function has no effect when there is no attempt to set a URL or to reconnect to a URL. The function is synchronous. When it aborted an `OpenUrl` or an automatic connection retry, it notifies the client with a synchronous `OnActionAborted` notification before completion and makes the `accessState` of the `currentUrl` equal to `AsNoAccess`.

Return Values

Table 948

Name	Description
`UH_OK`	The call led to an abort of a `OpenUrl` call or of an autonomous connection retry.
`UH_ERR_URLSRC_NOT_ABORTED`	The call did not abort since the action completed before the abort took place.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— If `urlSource.accessState == AsAccessing || urlSource.accessState == AsRetrying`
  — Cancel any action in an on-going `OpenUrl` or automatic re-connection.
  — Modify `urlSource.accessState`
  — Notify `OnActionAborted(*)`

Postcondition
— If `urlSource.accessState == AsAccessing || urlSource.accessState == AsRetrying`
  — `urlSource.accessState = AsNoAccess`

Remarks
— Only one of `OnAccessible`, `OnNotAccessible`, or `OnActionAborted` will be notified when the `AbortAction` is called that is intended to abort an `OpenUrl` or an autonomous reconnection call. The first two can be called when the following happens: before `AbortAction` inspects the `accessState`, either (1) the URL was found or the timeout occurred (for `OpenUrl`) or (2) the number of retries was reached or the connection was re-established (for the reconnection).

8.1.16.5.1.10 GetTotalSize

Signature
```
uhErrorCode_t GetTotalSize ( [out] pUInt32 pTotalSize );
```

Qualifiers
— synchronous

— thread-safe

Description
If the current URL is accessible and the total size of the data is known, this function returns that size. Otherwise it returns 0.

Parameters

Table 949

Name	Description
`pTotalSize`	Output parameter returning the total size of the current URL in bytes.

Return Values
Standard

Precondition
— True

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— If the current URL is accessible and the total size is known

```
Modify urlSource.totalSize
```

Postcondition
— *`*pTotalSize`* `== urlSource.totalSize`

8.1.16.5.1.11 GetTotalDuration

Signature
```
uhErrorCode_t GetTotalDuration ( [out] pUInt32 pTotalDuration );
```

Qualifiers
— synchronous

— thread-safe

Description
If the current URL is accessible and the total duration of the data is known, this function returns that duration. Otherwise it returns 0.

Parameters

Table 950

Name	Description
`pTotalDuration`	Output parameter returning the total duration of the current URL in milliseconds.

Return Values
Standard

Precondition
— True

Action
— If the current URL is accessible and the total duration is known
— `Modify urlSource.totalDuration`

Postcondition
— *`*pTotalDuration`* `== urlSource.totalDuration`

8.1.16.5.1.12 GetTrickControl

Signature
```
uhErrorCode_t GetTrickControl ( [out] uhIUrlSrcTrick **ppTrickControl );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Creates an object of type `ImageDecodingClient`, see subclause 8.1.8.4.2, when the server providing the URL and the platform have trick capabilities. It then returns an interface pointer of the `ImageDecodingClient` role if present, otherwise it returns a Null pointer.

Parameters

Table 951

Name	Description
`ppTrickControl`	Output parameter returning an interface pointer of `uhIUrlSrcTrick` interface if present, otherwise it returns a Null pointer.

Return Values
Standard

Precondition
`True`

Action
— If the server providing the URL and the platform have trick mode capabilities && `urlSource.pTrickControl == Null`
  — `urlSource.pTrickControl = create(trickControl)`

Postcondition
— *`*ppTrickControl`* == pointer to `uhIUrlSrcTrick` of `urlSource.pTrickControl`.

8.1.16.5.1.13 GetBufferFullness

Signature
```
uhErrorCode_t GetBufferFullness ( [out] pUInt32 pBufferFullness );
```

Qualifiers
— synchronous

— thread-safe

Description
If the current URL is accessible, this function returns the current buffer fullness level. Otherwise it returns 0.

Parameters

Table 952

Name	Description
`pBufferFullness`	Output parameter returning the current buffer fullness in % of the nominal buffer level for the current URL.

Return Values
Standard

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
— True

Action
— None

Postcondition
— **pBufferFullness* == `INT( 100 * urlSource.currentBufferLevel/nominalBufferLevel )`

Remark

The returned buffer fullness may be larger than 100%.

8.1.16.5.2 uhlUrlSrcNtf

Qualifiers
— callback

Description
This is a notification interface to be provided by clients of the URL Source logical component that wish to receive notifications of `UrlSrc` related events.

Interface ID
`uuid( 6921428F-2C90-43cc-9265-A202F3EBFFA0 )`

8.1.16.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhUrlSrc_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.16.5.2.2 OnAccessible

Signature
```
[async] Void OnAccessible ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
The `UrlSrc` calls this function when a requested URL has become accessible.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 953

Name	Description
cookie	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

8.1.16.5.2.3 OnNotAccessible

Signature

```
[async] Void OnNotAccessible (
    [in] UInt32                         cookie,
    [in] uhUrlSrc_NotAccessibleCause_t cause );
```

Qualifiers
None.

Description
The UrlSrc calls this function when the URL that was accessed is no longer accessible.

Parameters

Table 954

Name	Description
cookie	The value that was passed by the client at subscription time.
cause	The reason why the URL was not accessible.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

8.1.16.5.2.4 OnConnectionRetry

Signature
```
[async] Void OnConnectionRetry ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
The `UrlSrc` calls this function when the URL that was accessed is no longer accessible due to a lost connection. There are a number of autonomous retries and each retry causes such a notification. The time between retries is chosen by the platform. If not aborted (see AbortAction), the function is called `nrOfRetries` times before an `OnNotAccessible` is called.

Parameters

Table 955

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.16.5.2.5 OnEndOfRendering

Signature
```
[async] Void OnEndOfRendering ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
The `UrlSrc` calls this function when it detects that the end of a stream has been reached while playing. It indicates that the data is rendered and has left the platform.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This notification is signaled due to the following conditions (the notification will come after data has left the platform):

— The end of the stream was encountered while playing that stream (either forward or reverse). Note that it is irrelevant how the end was reached (for example using `SetPosition`) just the fact that the end is reached triggers the notification.

— An `OpenUrl` is called that interrupts a playing stream.

— An AbortAction is called that interrupts an `OpenUrl` or a connection retry, while a stream was being played.

— A `CloseUrl` is called while a stream was being played.

In case the output of `UrlSrc` is not really a streaming output but for example an image, this function is only called in the latter three cases; i.e. when the client does `OpenUrl`, `AbortAction`, or `CloseUrl`.

Parameters

Table 956

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.16.5.2.6 OnEndOfInput

Signature

```
[async] Void OnEndOfInput (
    [in] UInt32 cookie

);
```

Qualifiers
None.

Description
The `UrlSrc` calls this function while playing when the source of the data indicates that there is no more data. It does not mean that all data is rendered, but it indicates that a new `OpenUrl` can be performed without interrupting the current URL.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This notification is signaled only when an end of the input was encountered during normal streaming (playing forward or reverse in arbitrary speeds).

Parameters

Table 957

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
Thus a call to `OpenUrl`, `CloseUrl`, `AbortAction`, `SetPosition`, `StepForward`, or `StepBackward` will **not** cause an `OnEndOfInput` notification.

8.1.16.5.2.7 OnDataConsumed

Signature

```
[async] Void OnDataConsumed (
    [in] UInt32 cookie,
    [in] UInt32 bytesConsumed );
```

Qualifiers
None.

Description
The `UrlSrc` calls this function each time at least 1% of the data has been consumed. However, this function will not be called more than `urlSource.callbackRate` times per second.

This function will only be called if the total size is known.

Parameters

Table 958

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`bytesConsumed`	The number of bytes that have already been consumed of the data that is currently playing.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.16.5.2.8 OnPlayTimeChanged

Signature
```
[async] Void OnPlayTimeChanged (
    [in] UInt32 cookie,
    [in] UInt32 timePlayed );
```

Qualifiers
None.

Description
The `UrlSrc` calls this function each time at least 1% of the total duration has passed. However, this function will not be called more than `urlSource.callbackRate` times per second.

This function will only be called if the total duration is known.

Parameters

Table 959

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`timePlayed`	An approximation of the current position in milliseconds in the data that is currently playing.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks
After jumping in a variable bit rate stream, only an approximation of the time can be reported.

8.1.16.5.2.9 OnActionAborted

Signature
```
[async] Void OnActionAborted ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
The `UrlSrc` calls this function when the client has called `AbortAction` and indeed a `OpenUrl` or a connection retry was cancelled.

Parameters

Table 960

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.16.5.2.10 OnTrickControlLost

Signature
```
[async] Void OnTrickControlLost ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
The `UrlSrc` calls this function when a new `OpenUrl` looses the capability to do trick modes. This notification will only come when previously a `GetTrickControl` call was made that indeed returned an interface pointer and when this interface pointer is not released.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 961

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
The client is obliged to do a `Release` on the interface pointer he previously obtained with `GetTrickControl`. As an exception to the rule, it is allowed to do this `Release` from this notification function.

8.1.16.5.3 uhIUrlSrcTrick

Qualifiers
— None

Description
This interface provides basic trick mode functions that can typically be provided for streaming audio and video.

Interface ID
`uuid( 49B2546A-6850-4d85-B57A-9904AC41D548 )`

8.1.16.5.3.1 Pause

Signature
```
uhErrorCode_t Pause ( );
```

Qualifiers
— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Pauses playback.

Parameters
None.

Return Values
Standard

Precondition
`True`

Action
— If `urlSource.playState == PsPlaying`
  — Modify `urlSource.playState`

Postcondition
— If `urlSource.playState == PsPlaying`
  — `urlSource.playState == PsPaused`

8.1.16.5.3.2 Resume

Signature
```
uhErrorCode_t Resume ( );
```

Qualifiers
— synchronous

— single-threaded

Description
Resumes playback at the point in the stream where `Pause` halted it or from the position set by the last `SetPosition` call if `SetPosition` is called after `Pause`.

Parameters
None.

Return Values
Standard

Precondition
`True`

Action
— If `urlSource.playState == PsPaused`
  — Modify `urlSource.playState`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— If `urlSource.playState == PsPaused`
`urlSource.playState == PsPlaying`

8.1.16.5.3.3 SetPosition

Signature
```
uhErrorCode_t SetPosition (
    [in] UInt32 requestedTime,
    [in] uhUrlSrc_Rounding_t rounding,
    [out] pUInt32 pActualTime );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the position to the specified time. Returns the rounded position that was actually set.

Parameters

Table 962

Name	Description
`requestedTime`	The requested position (in milliseconds) in the current stream.
`rounding`	The rounding technique to be used.
`pActualTime`	Output parameter returning the actual position (in milliseconds) in the current stream.

Return Values
Standard

Precondition
`True`

Action
— Modify `trickControl.currentPosition`.

Postcondition
— `trickControl.currentPosition ==` rounding`(`*requestedTime*`)`
— `*`*pActualTime* `==` rounding`(`*requestedTime*`)`
— The position in the stream corresponds to **pActualTime* (in milliseconds).

Remarks
— There are restrictions on how accurate the position is from which playback resumes after a jump. For video, the previous I-picture has to be located for example. From there, playing can resume. For a variable bit rate stream, only an approximation of the position to jump to can be determined.

— Jumping to a time later than the total duration will result in a jump to the end of the stream.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.5.3.4 GetPosition

Signature
```
uhErrorCode_t GetPosition ( [out] pUInt32 pTime );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current position in the stream that is playing.

Parameters

Table 963

Name	Description
`pTime`	Output parameter returning the current position in the stream that is playing.

Return Values
Standard

Precondition
`True`

Action
None.

Postcondition
— *`*pTime == trickControl.currentPosition`*

8.1.16.5.3.5 GetSpeedRange

Signature
```
uhErrorCode_t GetSpeedRange (
    [out] pBool  pReverseSupp,
    [out] pUInt8 pMinSpeedNum,
    [out] pUInt8 pMinSpeedDenom,
    [out] pUInt8 pMaxSpeedNum,
    [out] pUInt8 pMaxSpeedDenom );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns (1) whether reverse speeds are allowed and (2) the lower and upper bounds of the speed that can be handled for the content identified by the URL. The bounds are determined as follows: `minSpeed = (*pMinSpeedNum/*pMinSpeedDenom)` and `maxSpeed = (*pMaxSpeedNum/*pMaxSpeedDenom)`.

Parameters

Table 964

Name	Description
`pReverseSupp`	Output parameter returning whether reverse speeds are supported.
`pMinSpeedNum`	Output parameter returning the numerator part of the lower limit of the playback speed.
`pMinSpeedDenom`	Output parameter returning the denominator part of the lower limit of the playback speed.
`pMaxSpeedNum`	Output parameter returning the numerator part of the upper limit of the playback speed.
`pMaxSpeedDenom`	Output parameter returning the denominator part of the upper limit of the playback speed.

Return Values
Standard

Precondition
`True`

Action
None.

Postcondition
— `*pReverseSupp == trickControl.actualReverseSupp`
— `*pMinSpeedNum == trickControl.actualMinSpeedNum`
— `*pMinSpeedDenom == trickControl.actualMinSpeedDenom`
— `*pMaxSpeedNum == trickControl.actualMaxSpeedNum`
— `*pMaxSpeedDenom == trickControl.actualMaxSpeedDenom`

8.1.16.5.3.6 SetSpeed

Signature
```
uhErrorCode_t SetSpeed (
    [in] Int8    requestedSpeedNum,
    [in] UInt8   requestedSpeedDenom,
    [out] pInt8  pActualSpeedNum,
    [out] pUInt8 pActualSpeedDenom );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the playback speed as follows: `requestedSpeed = (requestedSpeedNum/requestedSpeedDenom)`. Returns the speed that was actually set as follows: `actualSpeed = (actualSpeedNum/actualSpeedDenom)`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 965

Name	Description
`requestedSpeedNum`	The numerator part of the requested playback speed. When negative, the playback will be reverse.
`requestedSpeedDenom`	The denominator part of the requested playback speed.
`pActualSpeedNum`	Output parameter returning the numerator part of the actual playback speed. When negative, the playback will be reverse.
`pActualSpeedDenom`	Output parameter returning the denominator part of the actual playback speed.

Return Values
Standard

Precondition
— `requestedSpeedDenom != 0`
— If `trickControl.reverseSupp == True`
  — `(trickControl.actualMinSpeedNum/trickControl.actualMinSpeedDenom)` <=
    `(abs(`*requestedSpeedNum*`)/`*requestedSpeedDenom*`)` <=
    `(trickControl.actualMaxSpeedNum/trickControl.actualMaxSpeedDenom)`
— Else
  — `(trickControl.actualMinSpeedNum/trickControl.actualMinSpeedDenom)` <=
    `(`*requestedSpeedNum*`/`*requestedSpeedDenom*`)` <=
    `(trickControl.actualMaxSpeedNum/trickControl.actualMaxSpeedDenom)`

Action
— Modify `trickControl.requestedSpeedNum`
— Modify `trickControl.requestedSpeedDenom`
— Modify `trickControl.actualSpeedNum`
— Modify `trickControl.actualSpeedDenom`
— `trickControl.actualSpeedNum/trickControl.actualSpeedDenom` ==
  the speed closest to *requestedSpeedNum*`/`*requestedSpeedDenom*

Postcondition
— `trickControl.requestedSpeedNum ==` *requestedSpeedNum*
— `trickControl.requestedSpeedDenom ==` *requestedSpeedDenom*
— `*`*pActualSpeedNum* `== trickControl.actualSpeedNum`
— `*`*pActualSpeedDenom* `== trickControl.actualSpeedDenom`

8.1.16.5.3.7 GetSpeed

Signature
```
uhErrorCode_t GetSpeed (
    [out] pInt8  pRequestedSpeedNum,
    [out] pUInt8 pRequestedSpeedDenom,
    [out] pInt8  pActualSpeedNum,
    [out] pUInt8 pActualSpeedDenom );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the playback speed as follows: `requestedSpeed = (*`*pRequestedSpeedNum*`/*`*pRequestedSpeedDenom*`)` and the speed that was actually set as follows: `actualSpeed = (*`*pActualSpeedNum*`/*`*pActualSpeedDenom*`)`.

Parameters

Table 966

Name	Description
`pRequestedSpeedNum`	Output parameter returning the numerator part of the requested playback speed. When negative, the requested playback is reverse.
`pRequestedSpeedDenom`	Output parameter returning the denominator part of the requested playback speed.
`pActualSpeedNum`	Output parameter returning the numerator part of the actual playback speed. When negative, the actual playback is reverse.
`pActualSpeedDenom`	Output parameter returning the denominator part of the actual playback speed.

Return Values
Standard

Precondition
`True`

Action
None.

Postcondition
— `*`*pRequestedSpeedNum* `== trickControl.requestedSpeedNum`
— `*`*pRequestedSpeedDenom* `== trickControl.requestedSpeedDenom`
— `*`*pActualSpeedNum* `== trickControl.actualSpeedNum`
— `*`*pActualSpeedDenom* `== trickControl.actualSpeedDenom`

8.1.16.5.4 uhlUrlSrcTrickStep

Qualifiers
— None

Description
This interface provides specific trick mode functions that enable clients to step through a stream per access unit.

Interface ID
`uuid( 53383DD1-BA73-47a6-84F0-394D12E60B36 )`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.16.5.4.1 GetAvailStepModes

Signature
```
uhErrorCode_t GetAvailStepModes ( [out] puhUrlSrc_StepModes_t pAvailStepModes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the available step modes.

Parameters

Table 967

Name	Description
`pAvailStepModes`	Output parameter returning the available step modes.

Return Values
Standard

Precondition
`True`

Action
None.

Postcondition
— *`*pAvailStepModes == trickControl.actualAvailStepModes`*

8.1.16.5.4.2 StepForward

Signature
```
uhErrorCode_t StepForward ( );
```

Qualifiers
— synchronous

— single-threaded

Description
Performs playback of the next access unit. If there is no next access unit, a return value indicating this is returned. The function returns when there was a noticeable effect for the client.

When the connection was lost, the function immediately returns without waiting for retries.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters
None.

Return Values

Table 968

Name	Description
UH_OK	Indicates successful execution.
UH_ERR_URLSRC_END_OF_STREAM	The step cannot be performed because the end of the stream has been reached.
UH_ERR_URLSRC_CONNECTION_LOST	The step cannot be performed because the connection was lost.

Precondition
— uhUrlSrc_StepForward in trickControl.actualAvailStepModes
— urlSource.playState == PsPaused

Action
None.

Postcondition
None.

8.1.16.5.4.3 StepBackward

Signature
```
uhErrorCode_t StepBackward ( );
```

Qualifiers
— synchronous
— single-threaded

Description
Performs playback of the previous access unit. If there is no previous access unit, a return value indicating this is returned. The function returns when there was a noticeable effect for the client.

When the connection was lost, the function immediately returns without waiting for retries.

Parameters
None.

Return Values

Table 969

Name	Description
UH_OK	Indicates successful execution.
UH_ERR_URLSRC_END_OF_STREAM	The step cannot be performed because the begin of the stream has been reached.
UH_ERR_URLSRC_CONNECTION_LOST	The step cannot be performed because the connection was lost.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition

— `uhUrlSrc_StepBackward` in `trickControl.actualAvailStepModes`

`urlSource.playState == PsPaused`

Action

None.

Postcondition

None.

8.1.17 UhlVbiSlice

8.1.17.1 Concepts

8.1.17.1.1 VBI Slicing

The input for the VBI data capturing unit (or VBI Slicer) is video lines (e.g. ITU656). Data lines of an analog video input stream are lines 6 through 23 inclusive and lines 318 through 335 inclusive for the PAL and SECAM standards, and lines 10 through 21 inclusive and lines 273 through 284 inclusive for the NTSC standard. Video lines of the analog video input stream can also carry VBI data, e.g. Gemstar for the NTSC standard can use lines 22 through 25 inclusive and lines 285 through 288 inclusive.

The VBI Slicer extracts the VBI data from the VBI lines and passes it on to the client. Data of several VBI data types can be extracted. The variety of VBI data types (WST, CC, VPS, WSS or Gemstar) that can be extracted is platform specific.

The VBI Slicer can notify the client whenever new data of a video frame has been sliced, see Figure 98.

Each VBI standard has its own unique way of modulating (e.g. data rate, framing code) the bits on the input signal. Each VBI standard also defines the number of valid bytes (excluding header) that are sent per line. For example, VPS sends 13 bytes on line 16. WSS sends 14 bits on line 23 for 625-line PAL and SECAM systems, and sends 14 bits of data on lines 20 and 283 for 525-line NTSC systems.

Two independent streams of CC data are present in the analog video input stream: one in the odd field, the other one in the even field. The captured CC data from both odd and even fields will be sent to the client at the same time.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

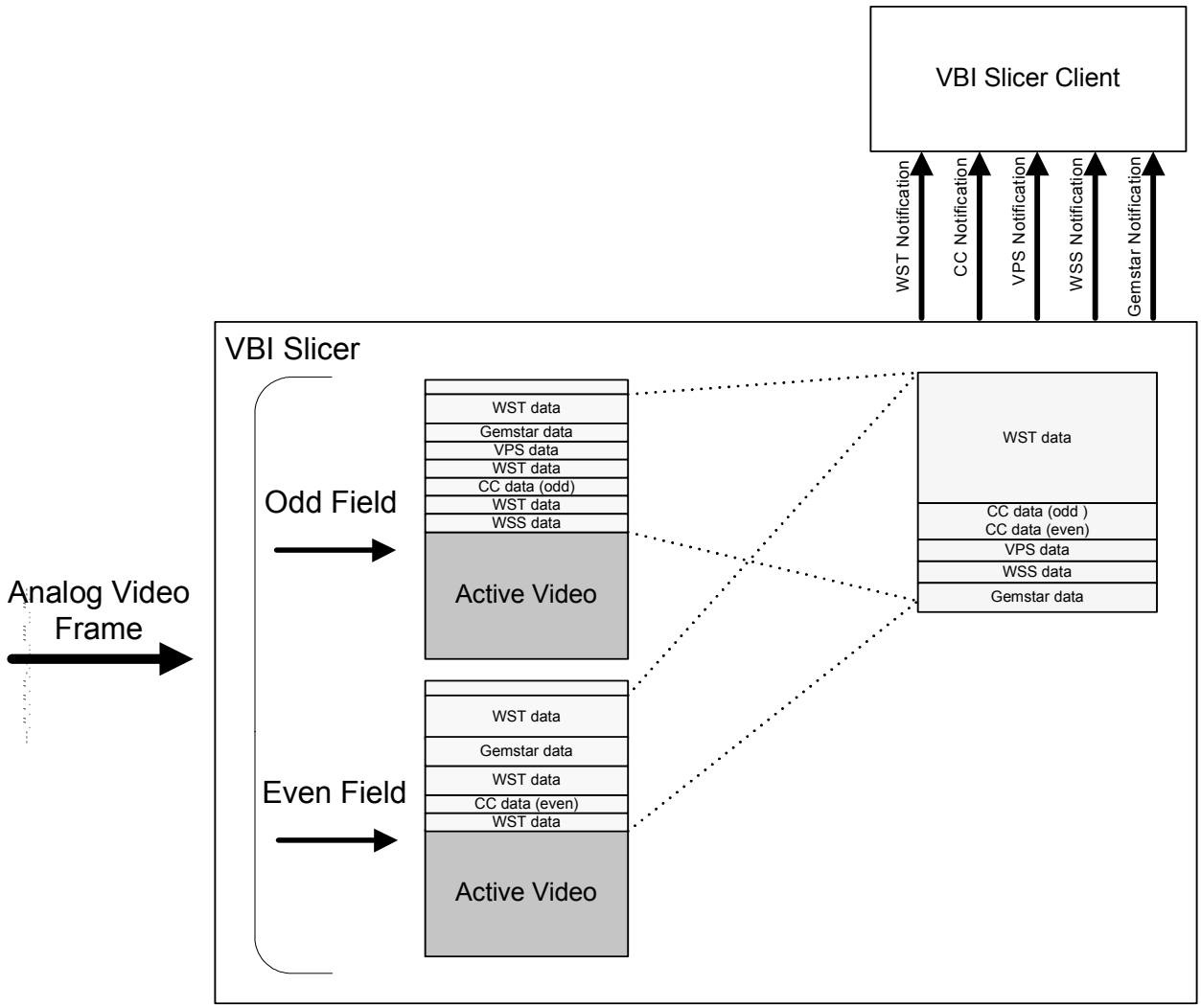


Figure 98 — Behavior of the VBI Slicer

The VBI Slicer extracts the VBI data present in the odd and even fields of a frame. When WST, CC, VPS, WSS and Gemstar data of a frame is sliced, each type of VBI data is notified separately to the client(s).

8.1.17.2 Types & Constants

8.1.17.2.1 Public Types & Constants

8.1.17.2.1.1 Error Codes

Signature

```
const uhErrorCode_t UH_ERR_VBISLICE_LINES_NOT_SET = 0x00000801;
```

Qualifiers

— error-codes

Description

The non-standard error codes that can be returned by functions of this logical component.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Constants

Table 970

Name	Description
UH_ERR_VBISLICE_LINES_NOT_SET	This error is specific to the VBI Slicer component and is only used in its interface uhIVbiSliceAutoSlice. This error is returned when the VBI Slicer cannot slice simultaneously all the specified data types on (at least one of) the given VBI line(s), or when no more VBI lines can be enabled, or when at least one VBI line cannot be disabled.

8.1.17.2.1.2 uhVbiSlice_Notification_t

Signature

```
typedef enum _uhVbiSlice_Notification_t {
    uhVbiSlice_SubscriptionChanged = 0x00000001,
    uhVbiSlice_FrameData           = 0x00000002,
    uhVbiSlice_FrameNoData         = 0x00000004,
    uhVbiSlice_Invalid             = 0x00000008
} uhVbiSlice_Notification_t, *puhVbiSlice_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interfaces uhIVbiSliceWstNtf, uhIVbiSliceCcNtf, uhIVbiSliceVpsNtf, uhIVbiSliceWssNtf and uhIVbiSliceGemstarNtf. The values can be used to individually turn on and off the notifications.

Values

Table 971

Name	Description
uhVbiSlice_SubscriptionChanged	Represents OnSubscriptionChanged, which occurs in all five notification interfaces.
uhVbiSlice_FrameData	Represents OnWstData, OnCcData, OnVpsData, OnWssData and OnGemstarData.
uhVbiSlice_FrameNoData	Represents OnWstNoData, OnCcNoData, OnVpsNoData, OnWssNoData and OnGemstarNoData.
uhVbiSlice_Invalid	Represents OnWstInvalid, OnCcInvalid, OnVpsInvalid, OnWssInvalid and OnGemstarInvalid.

8.1.17.2.1.3 uhVbiSlice_Notifications_t

Signature

```
typedef Int32 uhVbiSlice_Notifications_t, *puhVbiSlice_Notifications_t;
```

Qualifiers

— enum-set

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`VbiSlice_Notification_t`. These values are used in the `subscribe` and `unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.1.17.2.1.4 uhVbiSlice_AllNotifications

Signature
```
const uhVbiSlice_Notifications_t uhVbiSlice_AllNotifications = 0x0000000F;
```

Qualifiers
None.

Description
Defines the set of all values of type uh`VbiSlice_Notification_t`. This constant can be used in calls of the `subscribe` and `unsubscribe` functions to enable or disable all event notifications, respectively.

8.1.17.2.1.5 uhVbiSlice_Wst_t

Signature
```
typedef enum _uhVbiSlice_Wst_t {
    uhVbiSlice_EuroTeletext =  0,
    uhVbiSlice_UsTeletext   =  1,
    uhVbiSlice_Teletext     =  2,
    uhVbiSlice_Nabts        =  3,
    uhVbiSlice_Moji         =  4
} uhVbiSlice_Wst_t, *puhVbiSlice_Wst_t;
```

Qualifiers
None.

Description
Values of this type represent the different types of data standards for WST.

Values

Table 972

Name	Description
`uhVbiSlice_EuroTeletext`	Teletext EuroWST, CCST (Chinese); lines 6 through 22 inclusive and lines 318 through 335 inclusive; 42 bytes/line.
`uhVbiSlice_UsTeletext`	US Teletext (WST); lines 14 through 18 inclusive and lines 277 through 281 inclusive; 34 bytes/line.
`uhVbiSlice_Teletext`	Teletext; 42 bytes/line.
`uhVbiSlice_Nabts`	North-American Broadcast-Text-System (not for PAL); any combination of lines from line 10 to 20 inclusive and from line 273 to 283 inclusive; 36 bytes/line.
`uhVbiSlice_Moji`	Japanese Teletext; 35 bytes/line.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.2.1.6 uhVbiSlice_Cc_t

Signature

```
typedef enum _uhVbiSlice_Cc_t {
    uhVbiSlice_EuroClosedCaption =  0,
    uhVbiSlice_UsClosedCaption   =  1
} uhVbiSlice_Cc_t, *puhVbiSlice_Cc_t;
```

Qualifiers
None.

Description
Values of this type represent the different types of data standards for CC.

Values

Table 973

Name	Description
uhVbiSlice_EuroClosedCaption	European Closed Caption; line 22 and/or line 335; 2 bytes/line.
uhVbiSlice_UsClosedCaption	US Closed Caption; for 525-line NTSC systems, the US Closed Caption data can be present on line 21 and/or line 284; for 625-line Latin American PAL, US Closed Caption can also be present on a line other than line 21 and/or also on a line other than line 284; 2 bytes/line.

8.1.17.2.1.7 uhVbiSlice_Wss_t

Signature

```
typedef enum _uhVbiSlice_Wss_t {
    uhVbiSlice_EuroWss    = 0,
    uhVbiSlice_UsWss      = 1
} uhVbiSlice_Wss_t, *puhVbiSlice_Wss_t;
```

Qualifiers
None.

Description
Values of this type represent the different types of data standards for WSS.

Values

Table 974

Name	Description
uhVbiSlice_EuroWss	European WSS; line 23; 14 bits/line.
uhVbiSlice_UsWss	US WSS; line 20 and/or line 283; 14 bits of data/line (20 bits/line including CRC bits).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.2.1.8 uhVbiSlice_Gemstar_t

Signature

```
typedef enum _uhVbiSlice_Gemstar_t {
    uhVbiSlice_Gemstar1x      = 0,
    uhVbiSlice_Gemstar2x      = 1
} uhVbiSlice_Gemstar_t, *puhVbiSlice_Gemstar_t;
```

Qualifiers

None.

Description

Values of this type represent the different types of data standards for Gemstar.

Values

Table 975

Name	Description
uhVbiSlice_Gemstar1x	Gemstar 1X data; lines 10 through 25 inclusive and lines 273 through 288 inclusive; 2 bytes/line (only for US and Canada).
uhVbiSlice_Gemstar2x	Gemstar 2X data; lines 10 through 20 inclusive, lines 22 through 25 inclusive, lines 273 through 283 inclusive, lines 285 through 288 inclusive; 4 bytes/line (only for US and Canada).

8.1.17.2.1.9 uhVbiSlice_VbiData_t

Signature

```
typedef enum _uhVbiSlice_VbiData_t {
    uhVbiSlice_VbiNone               = 0x00000000,
    uhVbiSlice_VbiEuroTeletext       = 0x00000001,
    uhVbiSlice_VbiUsTeletext         = 0x00000002,
    uhVbiSlice_VbiTeletext           = 0x00000004,
    uhVbiSlice_VbiNabts              = 0x00000008,
    uhVbiSlice_VbiMoji               = 0x00000010,
    uhVbiSlice_VbiEuroClosedCaption  = 0x00000020,
    uhVbiSlice_VbiUsClosedCaption    = 0x00000040,
    uhVbiSlice_VbiEuroWss            = 0x00000080,
    uhVbiSlice_VbiUsWss              = 0x00000100,
    uhVbiSlice_VbiVps                = 0x00000200,
    uhVbiSlice_VbiGemstar1x          = 0x00000400,
    uhVbiSlice_VbiGemstar2x          = 0x00000800
} uhVbiSlice_VbiData_t, *puhVbiSlice_VbiData_t;
```

Qualifiers

— enum-element

Description

Values of this type represent the different types of VBI data standards.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 976

Name	Description
uhVbiSlice_VbiNone	No service, do not capture VBI data on that line.
uhVbiSlice_VbiEuroTeletext	Teletext EuroWST, CCST (Chinese); lines 6 through 22 inclusive and lines 318 through 335 inclusive; 42 bytes/line.
uhVbiSlice_VbiUsTeletext	US Teletext (WST); lines 14 through 18 inclusive and lines 277 through 281 inclusive; 34 bytes/line.
uhVbiSlice_VbiTeletext	Teletext; 42 bytes/line.
uhVbiSlice_VbiNabts	North-American Broadcast-Text-System (not for PAL); any combination of lines from line 10 to 20 inclusive and from line 273 to 283 inclusive; 36 bytes/line.
uhVbiSlice_VbiMoji	Japanese Teletext; 35 bytes/line.
uhVbiSlice_VbiEuroClosedCaption	European Closed Caption; line 22 and/or line 335; 2 bytes/line.
uhVbiSlice_VbiUsClosedCaption	US Closed Caption; for 525-line NTSC systems, the US Closed Caption data can be present on line 21 and/or line 284; for 625-line Latin American PAL, US Closed Caption can also be present on a line other than line 21 and/or also on a line other than line 284; 2 bytes/line.
uhVbiSlice_VbiEuroWss	European WSS; line 23; 14 bits/line.
uhVbiSlice_VbiUsWss	US WSS; line 20 and/or line 283; 14 bits of data/line (20 bits/line including CRC bits).
uhVbiSlice_VbiVps	Video Programming Service; line 16; 13 bytes/line.
uhVbiSlice_VbiGemstar1x	Gemstar 1X data; lines 10 through 25 inclusive and lines 273 through 288 inclusive; 2 bytes/line (only for US and Canada).
uhVbiSlice_VbiGemstar2x	Gemstar 2X data; lines 10 through 20 inclusive, lines 22 through 25 inclusive, lines 273 through 283 inclusive, lines 285 through 288 inclusive; 4 bytes/line (only for US and Canada).

8.1.17.2.1.10 uhVbiSlice_VbiDataSet_t

Signature

```
typedef Int32 uhVbiSlice_VbiDataSet_t, *puhVbiSlice_VbiDataSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of VBI data standards.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.2.2 Model Types & Constants

8.1.17.2.2.1 ValStat

Signature

```
typedef enum {
    ValStatInvalid = -2,
    ValStatUnknown = -1,
    ValStatKnown   =  1
} ValStat, *pValStat;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

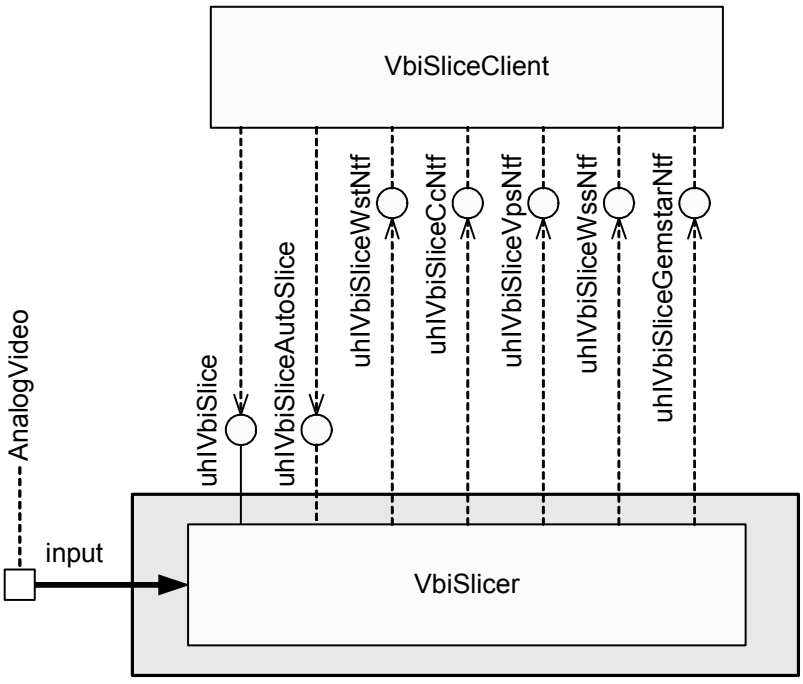
Values

Table 977

Name	Description
ValStatInvalid	The signal strength property is temporarily not available.
ValStatUnknown	The signal strength property is available, yet could not be determined.
ValStatKnown	The signal strength property is available and determined.

8.1.17.3 Logical Component

8.1.17.3.1 Interface-Role Model



© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Figure 99 — Interface-Role Model of the VBI Slicer

Figure 99 depicts the interface-role model of the VBI Slicer. It shows the subscription and notification interfaces and the roles the VBI Slicer and a VBI Slicer Client can have.

The VBI Slicer is represented by the `VbiSlicer` role.

The VBI Slicer provides facilities to subscribe/unsubscribe to notifications of events related to the extraction of WST, CC, VPS, WSS, and Gemstar data through its `uhIVbiSlice` interface. It can also provide functionality to enable and disable automatic extraction of VBI data through its `uhIVbiSliceAutoSlice` interface. Client(s) (represented by the `VbiSliceClient` role) can optionally provide one or more callback interfaces (`uhIVbiSliceWstNtf`, `uhIVbiSliceCcNtf`, `uhIVbiSliceVpsNtf`, `uhIVbiSliceWssNtf`, `uhIVbiSliceGemstarNtf`). These interfaces contain functions that will be called by the VBI Slicer to notify events.

8.1.17.3.2 Diversity

8.1.17.3.2.1 Provided Interfaces

Table 978

Role	Interface	Presence
`VbiSlicer`	`uhIvbiSlice`	mandatory
`VbiSlicer`	`uhIVbiSliceAutoSlice`	optional
`VbiSliceClient`	`uhIVbiSliceWstNtf`	optional
`VbiSliceClient`	`uhIVbiSliceCcNtf`	optional
`VbiSliceClient`	`uhIVbiSliceVpsNtf`	optional
`VbiSliceClient`	`uhIVbiSliceWssNtf`	optional
`VbiSliceClient`	`uhIVbiSliceGemstarNtf`	optional

8.1.17.3.2.2 Configurable Items

Table 979

Role	Attribute
`VbiSlicer`	`nrFramesNoData`

8.1.17.3.2.3 Constraints

```
nrFramesNoData >= 1
```

8.1.17.3.3 Instantiation

8.1.17.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 980

Type	Object	Multiplicity
`VbiSlicer`	`sli`	1

8.1.17.3.3.2 Initial State

— The VBI Slicer starts monitoring the VBI data lines, in its analog video input stream.
— The following constraints apply to the initial state of a logical component instance:
  — `sli.autoSlicingEnabled == True`
  — `sli.nrVbiLines == 0`
  — `sli.vbiLines == {}`
  — `sli.vbiTypes == {}`
  — `sli.wstStatus == ValStatInvalid`
  — `sli.ccStatus == ValStatInvalid`
  — `sli.vpsStatus == ValStatInvalid`
  — `sli.wssStatus == ValStatInvalid`
  — `sli.gemstarStatus == ValStatInvalid`

8.1.17.3.4 Execution Constraints

The interface provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier “thread-safe”.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.4 Roles

8.1.17.4.1 VbiSlicer

Signature

```
role VbiSlicer {
    const UInt8                     nrFramesNoData;
    Bool                            autoSlicingEnabled;
    UInt32                          nrVbiLines;
    setof{UInt32}                   vbiLines;
    setof{uhVbiSlice_VbiDataSet_t}  vbiTypes;
    ValStat                         wstStatus;
    ValStat                         ccStatus;
    ValStat                         vpsStatus;
    ValStat                         wssStatus;
    ValStat                         gemstarStatus;
}
```

Qualifiers

— root

Description

A VBI Slicer monitors an analog video input stream and extracts VBI data from it. The extraction of VBI data can be automatic or not. The possibility of disabling the automatic extraction of VBI data depends on the capabilities of the platform.

The VBI Slicer provides notifications:

— when it has captured the VBI data of a frame for the specific type(s) the client is subscribed to in case the VBI Slicer extracts VBI data automatically;

— when it has captured the VBI data of a frame from the requested line(s) with the requested data type(s) in case the VBI Slicer does not extract VBI data automatically;

— when the data lines corresponding to a specific data type (among others WST, CC, VPS, WSS and Gemstar) contain no data of this type for a specific number (`nrFramesNoData`) of frames.

When a client subscribes to the `OnWstData` notification, to the `OnWssData` notification or to the `OnGemstarData` notification, the VBI Slicer provides a notification each time it has monitored a frame (odd and even fields) of its analog video input stream, no matter whether it contains VBI data or not.

Independent Attributes

Table 981

Name	Description
`nrFramesNoData`	Indicates the number of consecutive frames without VBI data that determines that the stream of VBI data has ended.
`autoSlicingEnabled`	Indicates whether the VBI Slicer automatically detects and extracts the VBI data that can appear on each of the data lines of its analog video input stream. If True the VBI Slicer automatically detects and extracts the VBI data in its analog video input stream. If `False` the VBI Slicer does not automatically extract the VBI data in its analog video input stream.
`nrVbiLines`	The number of lines for which VBI data should to be sliced, i.e. the number of elements in arrays `vbiLines` and `vbiTypes` in case the VBI Slicer does not automatically extract the VBI data of its analog video input stream.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
vbiLines	The set of VBI line(s) that should be sliced in case the VBI Slicer does not automatically extract the VBI data of its analog video input stream. vbiLines[i] contains the VBI line to be sliced with VBI data type(s) vbiTypes[i].
vbiTypes	The set of VBI data type(s) that should be sliced in case the VBI Slicer does not automatically extract the VBI data of its analog video input stream. vbiTypes[i] contains the VBI data type(s) to be sliced for the VBI line vbiLines[i].
wstStatus	The validity status of the WST data.
ccStatus	The validity status of the CC data.
vpsStatus	The validity status of the VPS data.
wssStatus	The validity status of the WSS data.
gemstarStatus	The validity status of the Gemstar data.

Dependent Attributes
None.

Invariants

```
— nrFramesNoData >= 1
— If nrVbiLines == 0
   — vbiLines == {}
   — vbiTypes == {}
— For each i in {0..nrVbiLines -1}
   — vbiTypes[i] != uhVbiSlice_VbiNone
```

Instantiation
See subclause 8.1.17.3.3.

Streaming Behavior
The VBI Slicer continually monitors its analog video input stream and performs the following actions:

— extracting data from the data lines corresponding to any of the VBI data types (WST, CC, VPS, WSS or Gemstar) and raising an event when it has captured all VBI data of a frame in case the VBI Slicer extracts VBI data automatically;

  extracting the requested VBI data as defined in vbiTypes from the corresponding vbiLines data lines and raising an event when it has captured all requested VBI data of a frame in case the VBI Slicer does not extract VBI data automatically;

— raising an event when it has monitored a frame (odd and even fields) of its analog video input stream if client(s) are subscribed to the notification related to extraction of WST data (OnWstData), to the notification related to extraction of WSS data (OnWssData) or to the notification related to extraction of Gemstar data (OnGemstarData);

— raising an event when the data lines corresponding to a specific data type (among others WST, CC, VPS, WSS, and Gemstar) contain no data for nrFramesNoData consecutive frames since the last time they contained data of this type.

  The first action gives rise to the event notification defined by the OnYyyData (Yyy = Wst, Cc, Vps, Wss, Gemstar) notification function.

  The second action gives rise to the event notification defined by the OnYyyData (Yyy = Wst, Wss, Gemstar) notification function.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The third action gives rise to the event notification defined by the `OnYyyNoData` (`Yyy = Wst`, `Cc`, `Vps`, `Wss`, `Gemstar`) notification function.

See Figure 100 for an example where `nrFramesNoData == 3` in case of CC and VPS data. See Figure 101 for an example where `nrFramesNoData == 3` in case of WST, WSS, and Gemstar data.

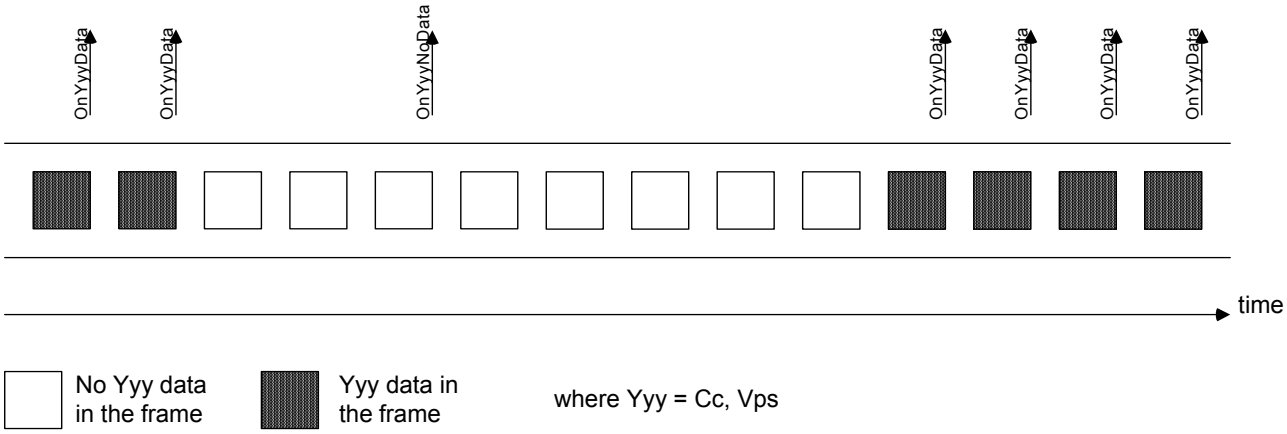


Figure 100 — Events Raised by the VBI Slicer in case of CC and VPS data.

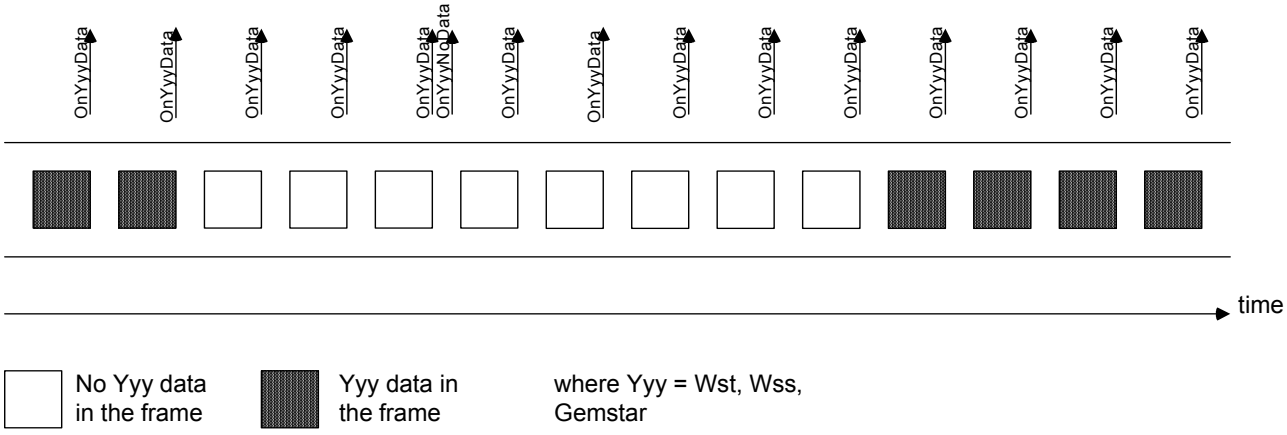


Figure 101 — Events Raised by the VBI Slicer in case of WST, WSS and Gemstar data.

Active Behavior

The VBI Slicer has standard notification behavior (see `uhIVbiSliceWstNtf`, `uhIVbiSliceCcNtf`, `uhIVbiSliceVpsNtf`, `uhIVbiSliceWssNtf and uhIVbiSliceGemstarNtf`).

Remarks

— When the extraction of VBI data is not automatic (`autoSlicingEnabled` is `False`) the VBI Slicer starts passing on VBI data to a client only after this client has defined the `vbiTypes` data types to extract from the corresponding `vbiLines` data lines and also only after this client has subscribed to the notifications of the type `OnYyyData`, where `Yyy` are the types in `vbiTypes`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— When the extraction of VBI data is automatic (`autoSlicingEnabled` is `True`) the VBI Slicer starts passing on VBI data to a client only after this client has subscribed to the notifications of the type `OnYyyData`.

— The modification of the extraction of VBI data from automatic to non-automatic, or from non-automatic to automatic, does not have any influence on the set of VBI lines which is stored in the `sli.vbiLines` attribute of the VBI Slicer and on the set of VBI types which is stored in the `sli.vbiTypes` attribute of the VBI Slicer.

8.1.17.4.2 VbiSliceClient

Signature
```
role VbiSliceClient {}
```

Qualifiers
— actor

Description
A VBI Slicer Client represents a user of the `uhIVbiSlice` and `uhIVbiSliceAutoSlice` interfaces. No constraints apply to the behavior of the VBI Slicer Client other than those implied by Figure 99.

8.1.17.5 Interfaces

8.1.17.5.1 uhIVbiSlice

Qualifiers
None.

Description
This interface of a VBI Slicer provides facilities to subscribe/unsubscribe to notifications of events related to the extraction of WST, CC, VPS, WSS and Gemstar data.

Interface ID
```
uuid( 0D1F574C-6643-407b-B505-1E3E269AC1C5 )
```

8.1.17.5.1.1 WstSubscribe

Signature
```
uhErrorCode_t WstSubscribe (
    [in] uhIVbiSliceWstNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.1.2 WstUnsubscribe

Signature

```
uhErrorCode_t WstUnsubscribe (
    [in] uhIVbiSliceWstNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.17.5.1.3 CcSubscribe

Signature

```
uhErrorCode_t CcSubscribe (
    [in] uhIVbiSliceCcNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— subscribe-function

8.1.17.5.1.4 CcUnsubscribe

Signature

```
uhErrorCode_t CcUnsubscribe (
    [in] uhIVbiSliceCcNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.17.5.1.5 VpsSubscribe

Signature

```
uhErrorCode_t VpsSubscribe (
    [in] uhIVbiSliceVpsNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— subscribe-function

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.1.6 VpsUnsubscribe

Signature

```
uhErrorCode_t VpsUnsubscribe (
    [in] uhIVbiSliceVpsNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.17.5.1.7 WssSubscribe

Signature

```
uhErrorCode_t WssSubscribe (
    [in] uhIVbiSliceWssNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— subscribe-function

8.1.17.5.1.8 WssUnsubscribe

Signature

```
uhErrorCode_t WssUnsubscribe (
    [in] uhIVbiSliceWssNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.17.5.1.9 GemstarSubscribe

Signature

```
uhErrorCode_t GemstarSubscribe (
    [in] uhIVbiSliceGemstarNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— subscribe-function

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.1.10 GemstarUnsubscribe

Signature

```
uhErrorCode_t GemstarUnsubscribe (
    [in] uhIVbiSliceGemstarNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.1.17.5.2 uhIVbiSliceAutoSlice

Qualifiers

None.

Description

This interface of a VBI Slicer provides functionality to enable and disable automatic extraction of VBI data from the data lines of the analog video input stream. It also provides functionality to set the VBI data types to capture per data line in case the extraction of VBI data is not automatic.

Interface ID

```
uuid( A0D11BAB-D693-45c1-8BAD-A58B37C714E5 )
```

8.1.17.5.2.1 GetAutoSlicingEnabled

Signature

```
uhErrorCode_t GetAutoSlicingEnabled ( [out] pBool pEnabled );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether the VBI Slicer automatically extracts the VBI data from its analog video input stream.

Parameters

Table 982

Name	Description
`pEnabled`	Output parameter returning whether the VBI Slicer automatically extracts the VBI data from its analog video input stream. If `True`, the VBI Slicer automatically extracts the VBI data from its analog video input stream. If `False`, the VBI Slicer does not automatically extract the VBI data from its analog video input stream.

Return Values

Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
`*pEnabled == sli.autoSlicingEnabled`

Remarks

— When the extraction of the VBI data is automatic, the VBI Slicer extracts data from any data line corresponding to any of the VBI data types (WST, CC, VPS, WSS or Gemstar) irrespective of the requested data type(s) in `sli.vbiTypes` and irrespective of the requested line(s) in `sli.vbiLines`. The VBI Slicer raises an event for each of the specific types the client is subscribed to when it has sliced the VBI data of a frame if data of such a specific type has been sliced.

— When the extraction of the VBI data is not automatic, the VBI Slicer only extracts the requested VBI data with data type as defined in `sli.vbiTypes` from the requested `sli.vbiLines` data lines, if data is present. The VBI Slicer raises an event for each of the specific types the client is subscribed to when it has sliced the VBI data of a frame if data of such a specific type has been sliced.

8.1.17.5.2.2 EnableAutoSlicing

Signature
```
uhErrorCode_t EnableAutoSlicing ( [in] Bool enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets whether the VBI Slicer should automatically extract the VBI data from its analog video input stream.

Parameters

Table 983

Name	Description
`enable`	The new setting indicating whether the VBI Slicer should automatically extract the VBI data from its analog video input stream. If `True`, the VBI Slicer automatically extracts the VBI data from its analog video input stream. If `False`, the VBI Slicer does not automatically extract the VBI data from its analog video input stream.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `sli.autoSlicingEnabled`

Postcondition
`sli.autoSlicingEnabled == enable`

Remarks

— The modification of the extraction of VBI data from automatic to non-automatic, or from non-automatic to automatic, does not have any influence on the set of VBI lines which is stored in the `sli.vbiLines` attribute of the VBI Slicer and on the set of VBI types which is stored in the `sli.vbiTypes` attribute of the VBI Slicer.

8.1.17.5.2.3 GetVbiLines

Signature
```
uhErrorCode_t GetVbiLines (
    [in] UInt32 size,
    [out] pUInt32 pActNrVbiLines,
    [out, size_is(size)] UInt32* pLines,
    [out, size_is(size)] uhVbiSlice_VbiDataSet_t* pDataTypes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the VBI lines for which VBI data should be sliced in case the extraction of VBI data is not automatic. The VBI data types that should be sliced for each of these lines are also returned.

Parameters

Table 984

Name	Description
`size`	The number of lines and the number of set of types that the buffers pointed by `pLines` and `pDataTypes` can contain.
`pActNrVbiLines`	The actual number of lines for which VBI data should be sliced in case the extraction of VBI data is not automatic, i.e. the number of relevant elements in arrays `pLines` and `pDataTypes`. Only `pLines[0]` through `pLines[*pActNrVbiLines-1]` inclusive and `pDataTypes[0]` through `pDataTypes[*pActNrVbiLines-1]` inclusive are relevant elements in array `pLines` and `pDataTypes`.
`pLines`	Pointer to the first element of an array with the VBI line(s) that should be sliced in case the extraction of VBI data is not automatic. `pLines[i]` contains the VBI line to be sliced with VBI data type(s) `pDataTypes[i]`.
`pDataTypes`	Pointer to the first element of an array with the VBI data type(s) that should be sliced in case the extraction of VBI data is not automatic. `pDataTypes[i]` contains the VBI data type(s) to be sliced for the VBI line `pLines[i]`.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
- `*pActNrVbiLines == sli.nrVbiLines`
- If `sli.nrVbiLines <= size`
  - For each `i` in `{0..sli.nrVbiLines-1}`
  - `pLines[i] == sli.vbiLines[i]`
  - `pDataTypes[i] == sli.vbiTypes[i]`
- Else
  - For each `i` in `{0..size -1}`
    - `pLines[i] == sli.vbiLines[i]`
    - `pDataTypes[i] == sli.vbiTypes[i]`

8.1.17.5.2.4 SetVbiLines

Signature
```
uhErrorCode_t SetVbiLines (
    [in] UInt32 nrVbiLines,
    [in, size_is(nrVbiLines)] const UInt32* pLines,
    [in, size_is(nrVbiLines)] const uhVbiSlice_VbiDataSet_t* pDataTypes );
```

Qualifiers
- synchronous
- single-threaded

Description
Sets the VBI data type(s) that should be sliced on the specified VBI line(s) in case the extraAction of VBI data is not automatic.

Parameters

Table 985

Name	Description
`nrVbiLines`	The number of lines to set with the VBI data type(s) to be sliced in case the extraction of VBI data is not automatic, i.e. the number of elements in arrays `pLines` and `pDataTypes`.
`pLines`	Pointer to the first element of an array with the VBI line(s) to be sliced in case the extraction of VBI data is not automatic. `pLines[i]` contains the VBI line to be sliced with VBI data type(s) `pDataTypes[i]`.
`pDataTypes`	Pointer to the first element of an array with the VBI data type(s) to be sliced in case the extraction of VBI data is not automatic. `pDataTypes[i]` contains the VBI data type(s) to be sliced for the VBI line `pLines[i]`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values

Table 986

Value	Description
UH_OK	Indicates successful execution.
UH_ERR_VBISLICE_LINES_NOT_SET	Indicates that the specified data type(s) to be sliced on the specified VBI line(s) have not be set since the VBI Slicer cannot slice simultaneously all the specified data types on (at least one of) the given VBI line(s), or no more VBI lines can be enabled, or at least one VBI line cannot be disabled.

Precondition

True

Action

— For each i in {0..nrVbiLines -1}
  — If pLines[i] in sli.vbiLines
    — Let j be the UInt32 such that sli.vbiLines[j] == pLines[i]
    — If pDataTypes[i] == uhVbiSlice_VbiNone
      — Modify sli.nrVbiLines
      — Modify sli.vbiLines
      — Modify sli.vbiTypes
    — Else if pDataTypes[i] != sli.vbiTypes[j]
      — Modify sli.vbiTypes[j]
  — Else
    — If pDataTypes[i] != uhVbiSlice_VbiNone
      — Modify sli.nrVbiLines
      — Modify sli.vbiLines
      — Modify sli.vbiTypes

Postcondition

— For each i in {0..nrVbiLines -1}
  — If pLines[i] in sli.vbiLines
    — Let j be the UInt32 such that sli.vbiLines[j] == pLines[i]
      — If pDataTypes[i] == uhVbiSlice_VbiNone
        — sli.nrVbiLines == sli.nrVbiLines' - 1
        — pLines[i] not in sli.vbiLines
      — Else if pDataTypes[i] != sli.vbiTypes[j]
        — sli.vbiTypes[j] == pDataTypes[i]
  — Else
    — If pDataTypes[i] != uhVbiSlice_VbiNone
      — sli.nrVbiLines == sli.nrVbiLines' + 1
      — There is a j in {0.. sli.nrVbiLines-1} for which
        — sli.vbiLines[j] == pLines[i]
        — sli.vbiTypes[j] == pDataTypes[i]

Remarks

— Although a certain VBI data service can be set for a certain line, this does not mean that this service will be available on that line, i.e. that valid data for the service has been found. For example, the VBI Slicer can be set up to slice lines 10 through 15 for uhVbiSlice_VbiEuroTeletext but it might not be available on line 12.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.3 uhIVbiSliceWstNtf

Qualifiers
— callback

Description
This interface should be provided by each VBI Slicer Client that wants to subscribe to notifications of events related to WST data that occur in the VBI Slicer. The `WstSubscribe` function of `uhIVbiSlice` can be used to subscribe to these events.

Interface ID
`uuid( F05A8483-6AC7-411e-BF1D-0F64D7099876 )`

8.1.17.5.3.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.17.5.3.2 OnWstData

Signature
```
Void OnWstData (
    [in] UInt32 cookie,
    [in] uhVbiSlice_Wst_t type,
    [in] UInt32 nrLines,
    [in, size_is(nrLines)] const UInt32 *pLines,
    [in] UInt32 sizeFirstBuffer,
    [in] UInt32 sizeSecondBuffer,
    [in, size_is(sizeFirstBuffer)] const UInt8 *pFirstBuffer,
    [in, size_is(sizeSecondBuffer)] const UInt8 *pSecondBuffer );
```

Qualifiers
None.

Description
This function is called by the VBI Slicer each time it has monitored a frame (odd and even fields) of its analog video input stream. The passed pointers `pFirstBuffer` and `pSecondBuffer` point to the sliced WST data of the monitored frame. The VBI Slicer owns the storage space occupied by the data. The storage spaces, and therefore the WST data, are guaranteed to stay unmodified until the next call of this function returns.

--`,,```,,,,````-`-,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 987

Name	Description
cookie	The value that was passed by the VBI Slicer client at subscription time.
type	The specific type of the WST data.
nrLines	The number of lines of the monitored video frame for which WST data has been sliced.
pLines	Pointer to a buffer containing the line numbers for which WST data has been sliced. pLines contains first the line numbers of the WST data stored in pFirstBuffer followed by the line numbers of the WST data stored in pSecondBuffer. See Figure 102 for an example of the organization of pLines and how pLines relates to the data in pFirstBuffer and in pSecondBuffer. If nrLines is 0, the content of pLines is irrelevant.
sizeFirstBuffer	The size, in bytes, of valid WST data to which pFirstBuffer points. If nrLines is 0, sizeFirstBuffer is 0.
sizeSecondBuffer	The size, in bytes, of valid WST data to which pSecondBuffer points. If nrLines is 0, sizeSecondBuffer is 0.
pFirstBuffer	Pointer to a sizeFirstBuffer bytes buffer of valid WST data. If sizeFirstBuffer is 0, the content of pFirstBuffer is irrelevant.
pSecondBuffer	Pointer to a sizeSecondBuffer bytes buffer of valid WST data. If sizeSecondBuffer is 0, the content of pSecondBuffer is irrelevant.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

Remarks
— When nrLines equals 0, no WST data has been extracted from the monitored video frame.

— The number of bytes per line of the specific type of WST data is known (see subclause 8.1.17.2.1.5).

— There is no specific relationship between the WST data to which pFirstBuffer points, and the WST data contained in the odd field of the video frames. The WST data to which pFirstBuffer points can originate from the odd field or even field of a video frame. The same holds for pSecondBuffer.

— The first byte of pFirstBuffer and of pSecondBuffer corresponds to the first byte of WST data on the first video line containing valid WST data in the odd and even fields or the other way around.

— The line numbers in pLines (and therefore the related WST data to which pFirstBuffer and pSecondBuffer point) are organised in increasing order. The layout of the bytes follows the WST standard, see [32].

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— When `sizeFirstBuffer` equals 0, `pFirstBuffer` does not point to valid WST data. The same holds for `sizeSecondBuffer` with respect to `pSecondBuffer`.

— Two examples of the organization of the storage space and the relationship between `pFirstBuffer`, `pSecondBuffer` and `pLines` are shown in Figure 102. On the left, WST data (uh`VbiSlice_EuroTeletext`) is present on every line from line 6 to 22 inclusive and from line 318 to 335 inclusive. On the right, only line 326 contains WST data (uh`VbiSlice_EuroTeletext`). `pLines` contains first the line numbers of the WST data stored in `pFirstBuffer` followed by the line numbers of the WST data stored in `pSecondBuffer`.

— When the VBI Slicer extracts data with uh`VbiSlice_Moji` data type, the captured bits (from the Moji data line(s)) are constructed into bytes. A Moji data line contains 272 bits but is stored in 35 bytes; the first 2 bits are filled with zeros and the last 6 are undefined and should be ignored.

8.1.17.5.3.3 OnWstNoData

Signature

```
[async] Void OnWstNoData ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the VBI Slicer whenever it detects that the data lines contained no WST data for `nrFramesNoData` consecutive frames since the last time they contained WST data.

Parameters

Table 988

Name	Description
`cookie`	The value that was passed by the VBI Slicer client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

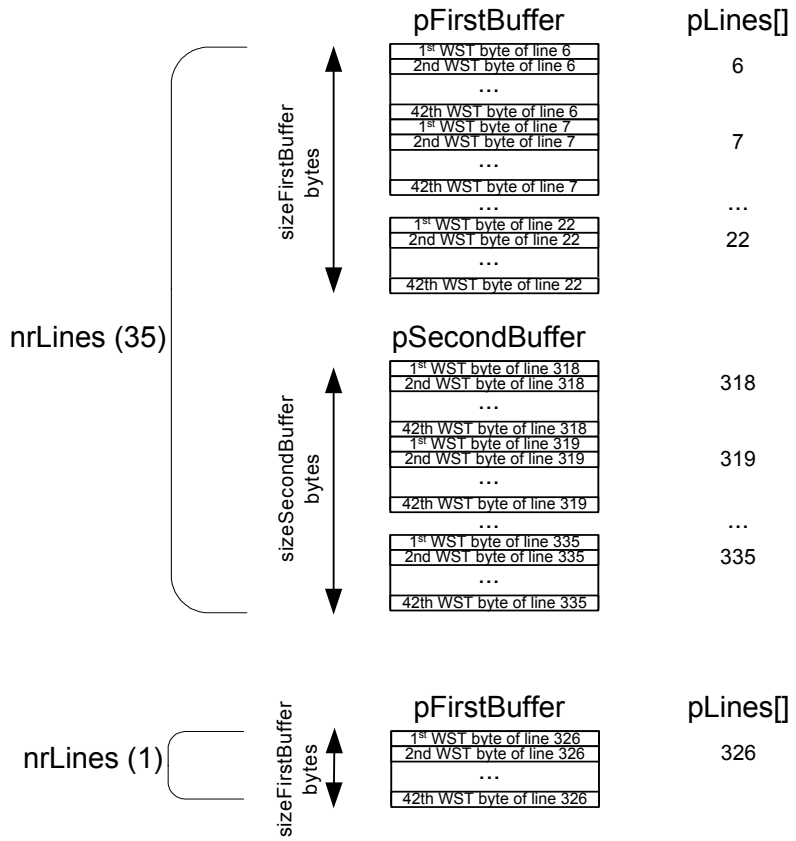


Figure 102 — Organization of captured WST data, showing two examples: for `uhVbiSlice_EuroTeletext` data (42 bytes/line). The first example (at the top) shows 35 lines where every line from line 6 to 22 inclusive and from line 318 to 335 inclusive contains data. The second example (at the bottom) shows a single line where only line 326 contains data.

8.1.17.5.3.4 OnWstInvalid

Signature

```
Void OnWstInvalid (
    [in] UInt32 cookie
);
```

Qualifiers
None.

Description
This function is called when the WST validity status has become invalid.

Parameters

Table 989

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.17.5.4 uhIVbiSliceCcNtf

Qualifiers
— callback

Description
This interface should be provided by each VBI Slicer Client that wants to subscribe to notifications of events related to CC data that occur in the VBI Slicer. The `CcSubscribe` function of `uhIVbiSlice` can be used to subscribe to these events.

Interface ID
`uuid( 63FED2B8-78C2-4007-952A-BD2B94002290 )`

8.1.17.5.4.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.17.5.4.2 OnCcData

Signature
```
[async] Void OnCcData (
[in] UInt32 cookie,
    [in] uhVbiSlice_Cc_t type,
    [in] UInt32 oLine,
    [in] UInt32 eLine,
    [in] UInt16 oField,
    [in] UInt16 eField );
```

Qualifiers
None.

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is called by the VBI Slicer whenever it has sliced CC data of a frame.

Parameters

Table 990

Name	Description
`cookie`	The value that was passed by the VBI Slicer client at subscription time.
`type`	The specific type of the CC data.
`oLine`	The line number for which CC data from the odd field has been sliced. If `oLine` is 0, the content of `oField` is irrelevant.
`eLine`	The line number for which CC data from the even field has been sliced. If `eLine` is 0, the content of `eField` is irrelevant.
`oField`	The CC data from the odd field. The bytes are stored as indicated in Figure 103.
`eField`	The CC data from the even field. The bytes are stored as indicated in Figure 103.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
— The bit layout of the notified data in `oField and eField follows the CC` standard; see [14] in the case of uh`VbiSlice_UsClosedCaption`. This is shown in Figure 103.

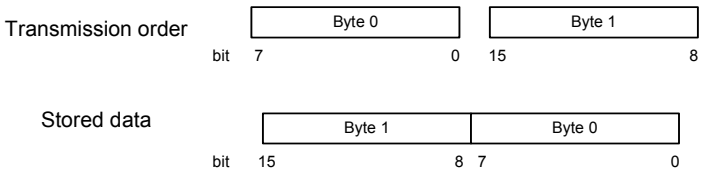


Figure 103 — The relationship of the bit layout of the captured CC data in `oField` and eField and the order of transmission of CC data in the analog video input stream.

— When this function is called by the VBI Slicer (i.e. when the VBI Slicer has sliced CC data of a frame), at least one of the `oLine` and `eLine` parameters is not equal to 0.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.4.3 OnCcNoData

Signature

```
[async] Void OnCcNoData (
    [in] UInt32 cookie,
    [in] Bool field );
```

Qualifiers
None.

Description
This function is called by the VBI Slicer whenever it detects that the CC data line of the field indicated by field contained no CC data for `nrFramesNoData` consecutive frames since the last time it contained CC data.

Parameters

Table 991

Name	Description
`cookie`	The value that was passed by the VBI Slicer client at subscription time.
`field`	Indicates the field whose CC data line contained no CC data for `nrFramesNoData` consecutive frames. It is `True` for the stream on even fields and `False` for the stream on odd fields.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
The parameter `field` is necessary since for CC data streams on odd and even fields are not related and thus can end at a different time.

8.1.17.5.4.4 OnCcInvalid

Signature

```
Void OnCcInvalid (
    [in] UInt32 cookie
);
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is called when the CC validity status has become invalid.

Parameters

Table 992

Name	Description
cookie	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.17.5.5 uhIVbiSliceVpsNtf

Qualifiers
— callback

Description
This interface should be provided by each VBI Slicer Client that wants to subscribe to notifications of events related to VPS data that occur in the VBI Slicer. The `VpsSubscribe` function of `uhIVbiSlice` can be used to subscribe to these events.

Interface ID
`uuid( A4940D72-01DA-4307-914F-0C70C23E8C18 )`

8.1.17.5.5.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.5.2 OnVpsData

Signature

```
Void OnVpsData (
    [in] UInt32 cookie,
    [in] UInt8 size,
    [in, size_is(size)] const UInt8 *pBuffer );
```

Qualifiers
None.

Description
This funAction is called by the VBI Slicer whenever it has sliced VPS data of a frame. The passed pointer points to the captured VPS data. The VBI Slicer owns the storage space occupied by the data. The storage space, and therefore the VPS data, is guaranteed to stay unmodified until the next call of this funAction returns.

Parameters

Table 993

Name	Description
`cookie`	The value that was passed by the VBI Slicer client at subscription time.
`size`	The size, in bytes, of valid VPS data.
`pBuffer`	Pointer to a size bytes buffer of valid data.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
— The number of bytes of VPS data is known: 13 bytes/line. The VPS data can be present on line 16 of the odd field.

— The first byte of the `pBuffer` corresponds to the first byte of valid VPS data on line 16.

— The layout of the bytes follows the VPS standard, see [30].

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.5.3 OnVpsNoData

Signature

```
[async] Void OnVpsNoData ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the VBI Slicer whenever it detects that the VPS data line contained no VPS data for `nrFramesNoData` consecutive frames since the last time it contained VPS data.

Parameters

Table 994

Name	Description
`cookie`	The value that was passed by the VBI Slicer Client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.17.5.5.4 OnVpsInvalid

Signature

```
Void OnVpsInvalid (
    [in] UInt32 cookie
);
```

Qualifiers
None.

Description
This function is called when the VPS validity status has become invalid.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 995

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.17.5.6 uhIVbiSliceWssNtf

Qualifiers
— callback

Description
This interface should be provided by each VBI Slicer Client that wants to subscribe to notifications of events related to WSS data that occur in the VBI Slicer. The `WssSubscribe` function of `uhIVbiSlice` can be used to subscribe to these events.

Interface ID
```
uuid( 23AD8822-CD3C-438f-8996-535AC88204E4 )
```

8.1.17.5.6.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.17.5.6.2 OnWssData

Signature
```
[async] Void OnWssData (
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32 cookie,
    [in] uhVbiSlice_Wss_t type,
    [in] Bool wssPresentOddField,
    [in] Bool wssPresentEvenField,
    [in] UInt16 dataOddField,
    [in] UInt16 dataEvenField,
    [in] UInt16 errorBitsOddField,
    [in] UInt16 errorBitsEvenField );
```

Qualifiers
None.

Description
This function is called by the VBI Slicer each time it has monitored a frame (odd and even fields) of its analog video input stream. For each field, information over the actual extraction of the WSS data is passed as well as the WSS data and the error bits if extracted.

Parameters

Table 996

Name	Description
cookie	The value that was passed by the VBI Slicer client at subscription time.
type	The specific type of the WSS data.
wssPresentOddField	Indicates whether WSS data from the odd field of the current frame has been extracted.
wssPresentEvenField	Indicates whether WSS data from the even field of the current frame has been extracted.
dataOddField	If wssPresentOddField is True, dataOddField contains the WSS data of the odd field of the current frame. The WSS bits are in bits 0..13. If wssPresentOddField is False, the content of dataOddField is irrelevant.
dataEvenField	If wssPresentEvenField is True, dataEvenField contains the WSS data of the even field of the current frame. The WSS bits are in bits 0..13. If wssPresentEvenField is False, the content of dataEvenField is irrelevant.
errorBitsOddField	If wssPresentOddField is True, errorBitsOddField contains the error bits accompanying the WSS data of the odd field of the current frame.When the type of the WSS data is uhVbiSlice_EuroWss, each bit of errorBitsOddField indicates, when it equals 1, that there is an error in the corresponding WSS bit of dataOddField.When the type of the WSS data is uhVbiSlice_UsWss, if errorBitsOddField is not equal to 0, it indicates that the CRC check on the data in dataOddField failed.If wssPresentOddField is False, the content of errorBitsOddField is irrelevant.
errorBitsEvenField	If wssPresentEvenField is True, errorBitsEvenField contains the error bits accompanying the WSS data of the even field of the current frame.When the type of the WSS data is uhVbiSlice_EuroWss, each bit of errorBitsEvenField indicates, when it equals 1, that there is an error in the corresponding WSS bit of dataEvenField.When the type of the WSS data is uhVbiSlice_UsWss, if errorBitsEvenField is not equal to 0, it indicates that the CRC check on the data in dataEvenField failed.If wssPresentEvenField is False, the content of errorBitsEvenField is irrelevant.

Return Values
None.

Precondition
True

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
— The number of bits of WSS data is known: 14 bits/line. The WSS data can be present only on line 23 of the odd field for 625-line PAL and SECAM systems. For 525-line NTSC systems, the WSS data can be present on lines 20 and 283.

— When the type of the WSS data is uh`VbiSlice_EuroWss`, the layout of the bits follows the WSS standard as described in [31]. The value of the remaining two bits in the `dataOddField` is unspecified.

— When the type of the WSS data is uh`VbiSlice_UsWss`, the layout of the bits follows bit assignment and usage as defined in [33]. The value of the remaining two bits in the `dataOddField` and `dataEvenField` is unspecified. Figure 104 depicts the layout of the bits in `dataOddField` and `dataEvenField`.

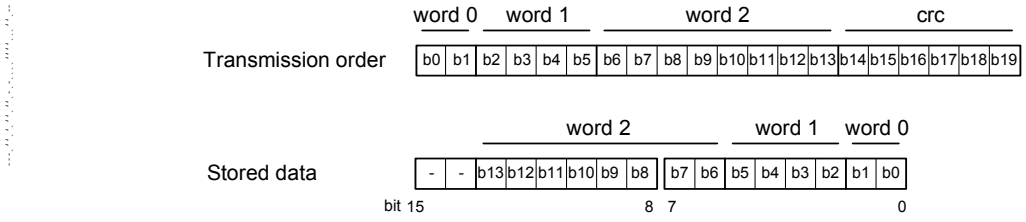


Figure 104 — The relationship of the bit layout of the captured 525-line WSS data in `dataOddField` and `dataEvenField` and the order of transmission of 525-line WSS data in the analog video input stream.

8.1.17.5.6.3 OnWssNoData

Signature
```
[async] Void OnWssNoData ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by the VBI Slicer whenever it detects that the WSS data line(s) contained no WSS data for `nrFramesNoData` consecutive frames since the last time it contained WSS data.

Parameters

Table 997

Name	Description
`cookie`	The value that was passed by the VBI Slicer Client at subscription time.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.1.17.5.6.4 OnWssInvalid

Signature
```
Void OnWssInvalid (
    [in] UInt32 cookie
);
```

Qualifiers
None.

Description
This function is called when the WSS validity status has become invalid.

Parameters

Table 998

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.7 uhIVbiSliceGemstarNtf

Qualifiers
— callback

Description
This interface should be provided by each VBI Slicer Client that wants to subscribe to notifications of events related to extraction of Gemstar data that occur in the VBI Slicer. The `GemstarSubscribe` function of `uhIVbiSlice` can be used to subscribe to these events.

Interface ID
`uuid( DE154F62-E874-40c2-A1E5-605AA856F0AE )`

8.1.17.5.7.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVbiSlice_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.1.17.5.7.2 OnGemstarData

Signature
```
[async] Void OnGemstarData (
    [in] UInt32 cookie,
    [in] UInt32 nrLines,
    [in, size_is(nrLines)] const UInt32 *pLines,
    [in, size_is(nrLines)] const uhVbiSlice_Gemstar_t *pTypes,
    [in] UInt32 size,
    [in, size_is(size)] const UInt8 *pBuffer );
```

Qualifiers
None.

Description
This function is called by the VBI Slicer each time it has monitored a frame (odd and even fields) of its analog video input stream. The passed pointer `pBuffer` points to the sliced Gemstar data of the monitored frame. The VBI Slicer owns the storage space occupied by the data. The storage spaces, and therefore the Gemstar data, are guaranteed to stay unmodified until the next call of this function returns.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 999

Name	Description
cookie	The value that was passed by the VBI Slicer Client at subscription time.
nrLines	The number of lines of the monitored video frame for which Gemstar data has been sliced.
pLines	Pointer to a buffer containing the line numbers for which Gemstar data has been sliced. pLines[i] contains the VBI line sliced with Gemstar data type pTypes[i]. If nrLines is 0, the content of pLines is irrelevant.
pTypes	Pointer to a buffer containing the Gemstar type for each line for which Gemstar data has been sliced. If nrLines is 0, the content of pTypes is irrelevant.
size	The size, in bytes, of valid Gemstar data to which pBuffer points. If nrLines is 0, size is 0.
pBuffer	Pointer to a 4 byte aligned buffer with size bytes of valid Gemstar data. pBuffer points to the first element of this data. If size is 0, the content of pBuffer is irrelevant.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

Remarks
— When nrLines equals 0, no Gemstar data has been extracted from the monitored video frame.

— Gemstar data are stored in a 4 byte aligned buffer, i.e. 4 bytes are used to stored data of the uhVbiSlice_Gemstar1x and of the uhVbiSlice_Gemstar2x type. In case of uhVbiSlice_Gemstar1x data, the first two bytes contain the sliced Gemstar 1x data, the content of the remaining two bytes is undefined and should be ignored.

— When the extraction of VBI data is not automatic (sli.autoSlicingEnabled == False), the first byte of pBuffer corresponds to the first byte of sliced Gemstar data on the first video line requested that contains valid Gemstar data of a requested Gemstar data type.

— When the extraction of VBI data is automatic (sli.autoSlicingEnabled == True), the first byte of pBuffer corresponds to the first byte of sliced Gemstar data on the first video line that contains valid Gemstar data.

— The line numbers in pLines (and therefore the related Gemstar data to which pBuffer points) are organised in increasing order.

— For each line number in pLines, the size, in bytes, of the data can be retrieved from the data type indicated in pTypes. See subclause 8.1.17.2.1.8.

— The layout of the bytes follows the Gemstar standard, see [34].

— An example of the general layout of the storage space is shown in Figure 105.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

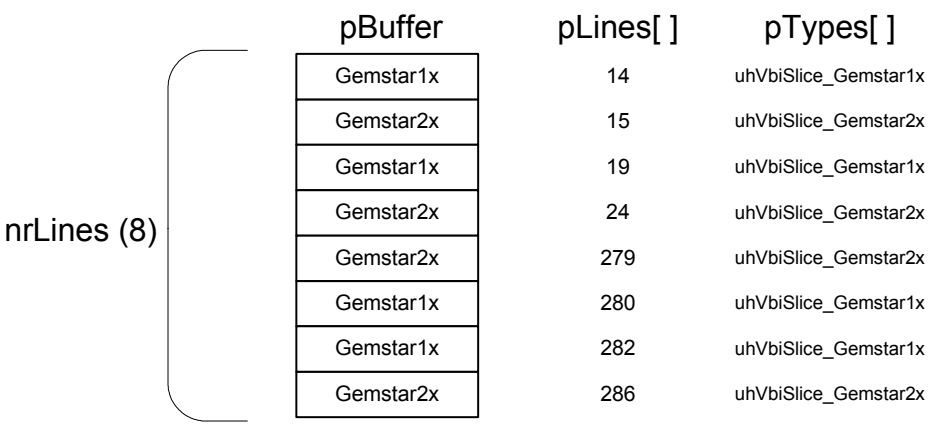


Figure 105 — General layout of captured Gemstar data.

8.1.17.5.7.3 OnGemstarNoData

Signature

```
[async] Void OnGemstarNoData (
    [in] UInt32 cookie,
    [in] uhVbiSlice_Gemstar_t gemstarType );
```

Qualifiers
None.

Description
This function is called by the VBI Slicer whenever it detects that the data lines contained no Gemstar data of the `gemstarType` type for `nrFramesNoData` consecutive frames since the last time they contained Gemstar data of this type.

Parameters

Table 1000

Name	Description
`cookie`	The value that was passed by the VBI Slicer Client at subscription time.
`gemstarType`	Indicates the Gemstar type for which data lines contained no data for `nrFramesNoData` consecutive frames since the last time they contained Gemstar data of this type.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.17.5.7.4 OnGemstarInvalid

Signature

```
Void OnGemstarInvalid (
    [in] UInt32 cookie
);
```

Qualifiers
None.

Description
This function is called when the Gemstar validity status has become invalid.

Parameters

Table 1001

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition

```
True
```

Action
— Any callback-compliant action.

Postcondition

```
True
```

8.2 Video processing and rendering

8.2.1 UhlAmbientLevel

8.2.1.1 Concepts

8.2.1.1.1 Ambient Light

Video improvement algorithms that automatically change settings like brightness and contrast, in order to get a better picture quality, can attain a better result if they have an indication of the amount of ambient light in the room where the television set is, since this ambient light has a big influence on the way such settings are perceived.

The Ambient Level logical component, described in this document, provides the client with means to make the amount of ambient light known to the platform (see Figure 106).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

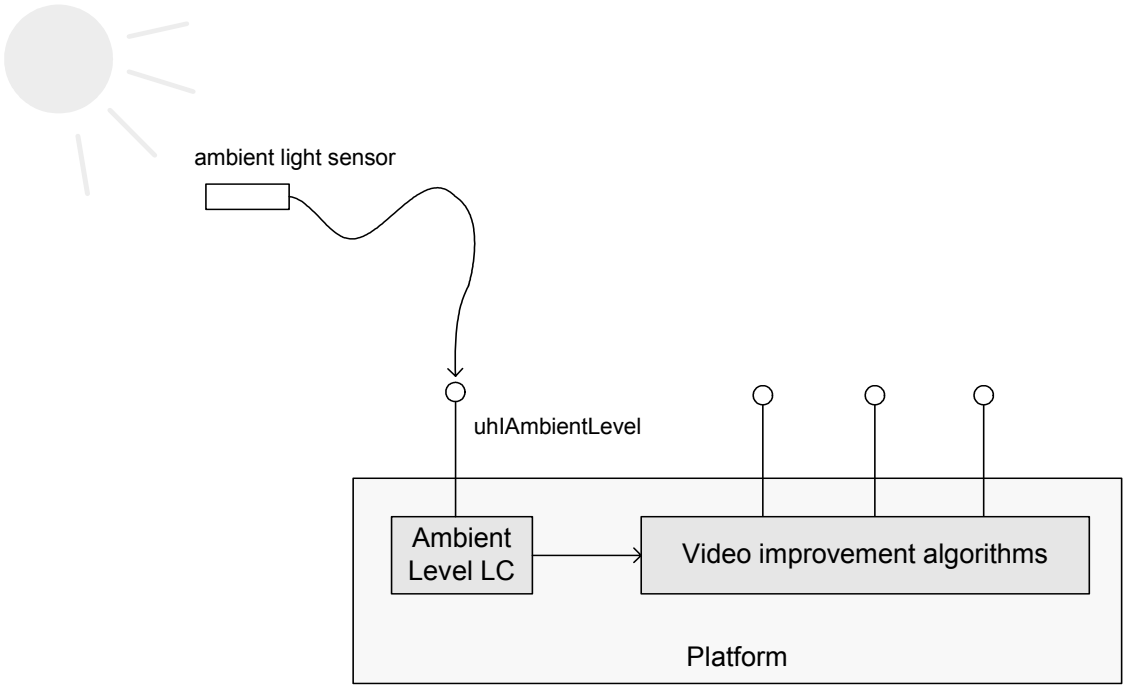


Figure 106 — The role of the Ambient Level LC in dealing with ambient light.

8.2.1.2 Types & Constants

8.2.1.2.1 Public Types & Constants

No additional types and constants are required for this logical component.

8.2.1.3 Logical Component

8.2.1.3.1 Interface-Role Model

The interface role-model is depicted in Figure 107.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

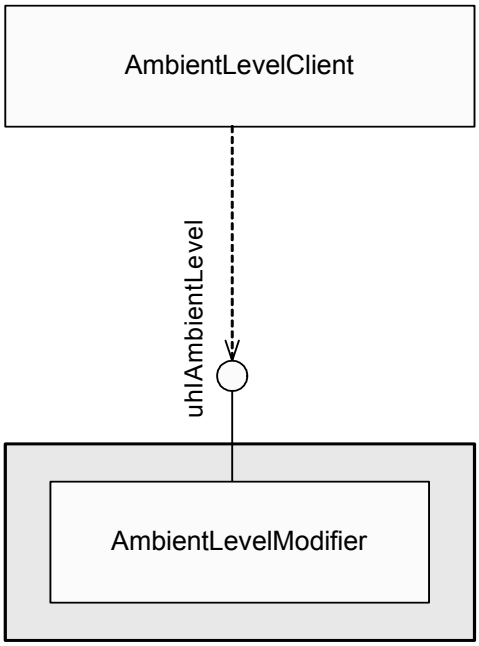


Figure 107 — Interface-Role Model of the Ambient Level logical component

The Ambient Level logical component (grey box) consists of the `AmbientLevelModifier` role. This role provides an interface (`uhIAmbientLevel`) for controlling the AL level. The control interface is meant for use by the client (AmbientLevelClient).

8.2.1.3.2 Diversity

8.2.1.3.2.1 Provided Interfaces

Table 1002

Role	Interface	Presence
`AmbientLevelModifier`	`uhIAmbientLevel`	mandatory

8.2.1.3.2.2 Configurable Items

Table 1003

Role	Attribute
`AmbientLevelModifier`	`minLevel`
`AmbientLevelModifier`	`maxLevel`

8.2.1.3.2.3 Constraints

— `maxLevel >= minLevel`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.1.3.3 Instantiation

8.2.1.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1004

Type	Object	Multiplicity
`AmbientLevelModifier`	`alm`	1

8.2.1.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— `alm.level == alm.minLevel`

8.2.1.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.2.1.4 Roles

8.2.1.4.1 AmbientLevelModifier

Signature

```
role AmbientLevelModifier {
    const UInt32              minLevel;
    const UInt32              maxLevel;
    UInt32                    level;
}
```

Qualifiers

— root

Description

The `AmbientLevelModifier` role controls the AL level which can be used by video improvement algorithms in the platform.

Independent Attributes

Table 1005

Name	Description
`minLevel`	The minimum value for the AL level.
`maxLevel`	The maximum value for the AL level.
`level`	The current value of the AL level.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes

None.

Invariants

— `minLevel <= level <= maxLevel`

Instantiation
See subclause 8.2.1.3.3.

Streaming Behavior
None.

Active Behavior
None.

8.2.1.4.2 AmbientLevelClient

Signature
`role AmbientLevelClient {}`

Qualifiers
— actor

Description
An `AmbientLevelClient` represents a user of the `uhIAmbientLevel` interface.

8.2.1.5 Interfaces

8.2.1.5.1 uhIAmbientLevel

Qualifiers
None.

Description
This interface controls the level of AL, which can be used by video improvement algorithms in the platform.

Interface ID
`uuid( 9431A895-0B70-4ed5-84F8-E23E351A037C )`

Remarks
In the following function specifications, the name `alm` will be used to refer to the provider of this interface; see subclause 8.2.1.3.3.1.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.1.5.1.1 GetLevel

Signature
```
uhErrorCode_t GetLevel (
    [out] pUInt32 pLevel );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current AL level.

Parameters

Table 1006

Name	Description
`pLevel`	Output parameter returning the current AL level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pLevel == alm.level`

8.2.1.5.1.2 SetLevel

Signature
```
uhErrorCode_t SetLevel (
    [in] UInt32 level);
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the AL level.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1007

Name	Description
`level`	The AL level to be set.

Return Values
Standard.

Precondition
`alm.minLevel <= level<= alm.maxLevel`

Action
— Modify `alm.level`

Postcondition
`alm.level == level`

GetLevelRange

Signature
```
uhErrorCode_t GetLevelRange (
    [out] pUInt32 pMinLevel,
    [out] pUInt32 pMaxLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum value for the AL level.

Parameters

Table 1008

Name	Description
`pMinLevel`	Output parameter returning the minimum value for the AL level.
`pMaxLevel`	Output parameter returning the maximum value for the AL level.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— **pMinLevel* == alm.minLevel
— *pMaxLevel == alm.maxLevel

8.2.2 UhlAnaVenc

8.2.2.1 Concepts

8.2.2.1.1 Analog Video Encoder

This document defines the Analog Video Encoder logical component (LC). The Analog Video Encoder LC gives control over the encoding of a digital video stream into an analog video signal. The encoding type can be CVBS or YC. The encoded analog video signal has a specified color system. This color system can be a variety of PAL, NTSC or SECAM.

8.2.2.2 Types & Constants

8.2.2.2.1 Public Types & Constants

8.2.2.2.1.1 Error Codes

None.

8.2.2.2.1.2 uhAnaVenc_EncodingType_t

Signature
```
typedef enum uhAnaVenc_EncodingType_t {
    uhAnaVenc_Cvbs     = 0x00000001,
    uhAnaVenc_Yc       = 0x00000002
} uhAnaVenc_EncodingType_t, *puhAnaVenc_EncodingType_t;
```

Qualifiers
— enum-element

Description
Describes the different encoding types.

Values

Table 1009

Name	Description
uhAnaVenc_Cvbs	CVBS encoding type.
uhAnaVenc_Yc	YC encoding type.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.2.2.1.3 uhAnaVenc_EncodingTypeSet_t

Signature
```
typedef Int32 uhAnaVenc_EncodingTypeSet_t, *puhAnaVenc_EncodingTypeSet_t;
```

Qualifiers
— enum-set

Description
Values of this type identify sets of encoding types, which are typically constructed by logical OR-ing values of type `uhAnaVenc_EncodingType_t`.

8.2.2.2.1.4 uhAnaVenc_ColorSystem_t

Signature
```
typedef enum _uhAnaVenc_ColorSystem_t {
    uhAnaVenc_PalM        = 0x00000001,
    uhAnaVenc_PalN        = 0x00000002,
    uhAnaVenc_PalBg       = 0x00000004,
    uhAnaVenc_NtscM       = 0x00000008,
    uhAnaVenc_Secam       = 0x00000010
} uhAnaVenc_ColorSystem_t, *puhAnaVenc_ColorSystem_t;
```

Qualifiers
— enum-element

Description
Describes the different color systems.

Values

Table 1010

Name	Description
`uhAnaVenc_PalM`	PAL-M color system.
`uhAnaVenc_PalN`	PAL-N color system.
`uhAnaVenc_PalBg`	PAL-BG color system.
`uhAnaVenc_NtscM`	NTSC-M color system.
`uhAnaVenc_Secam`	SECAM color system.

8.2.2.2.1.5 uhAnaVenc_ColorSystemSet_t

Signature
```
typedef Int32 uhAnaVenc_ColorSystemSet_t, *puhAnaVenc_ColorSystemSet_t;
```

Qualifiers
— enum-set

Description
Values of this type identify sets of color systems, which are typically constructed by logical OR-ing values of type `uhAnaVenc_ColorSystem_t`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.2.2.2 Model Types & Constants

None.

8.2.2.3 Logical Component

8.2.2.3.1 Interface-Role Model

Figure 108 depicts the interface-role model of the Analog Video Encoder LC. It shows the roles and interfaces the Analog Video Encoder LC and its client can have.

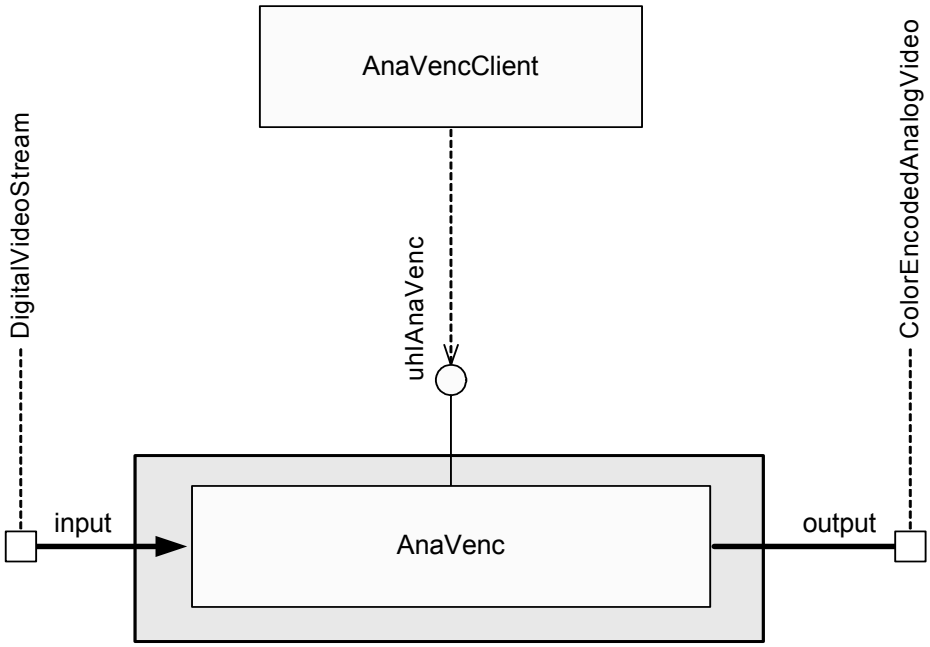


Figure 108 — Analog Video Encoder Interface-Role Model

The Analog Video Encoder LC (grey box) consists of the `AnaVenc` role. This role provides the interface `uhIAnaVenc` for encoding a digital video stream into an analog video signal. The control interface is meant for usage by the client `AnaVencClient`.

8.2.2.3.2 Diversity

8.2.2.3.2.1 Provided Interfaces

Table 1011

Role	Interface	Presence
`AnaVenc`	`uhIAnaVenc`	mandatory

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.2.3.2.2 Configurable Items

Table 1012

Role	Attribute
AnaVenc	encodingType
AnaVenc	supportedEncodingTypes
AnaVenc	colorSystem
AnaVenc	supportedColorSystems

8.2.2.3.2.3 Constraints

— See subclause *Invariants* in 7.2.1.4

8.2.2.3.3 Instantiation

8.2.2.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1013

Type	Object	Multiplicity
AnaVenc	anaVenc	1

8.2.2.3.3.2 Initial State

None.

8.2.2.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

All interfaces described for the Analog Video Encoding logical component are intended to be used by a single client. The single client nature of the Analog Video Encoding interfaces is exposed by the fact that any change in one of the analog video encoding method parameters does not result in a notification being generated. If multiple clients are acting on the Analog Video Encoding logical component, they might interfere with each other without further notice.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.2.4 Roles

8.2.2.4.1 AnaVenc

Signature

```
role AnaVenc {
    uhAnaVenc_EncodingType_t    encodingType;
    uhAnaVenc_EncodingTypeSet_t supportedEncodingTypes;
    uhAnaVenc_ColorSystem_t     colorSystem;
    uhAnaVenc_ColorSystemSet_t  supportedColorSystems;
}
```

Qualifiers

— root

Description

The `AnaVenc` controls which encoding type and color system is used for encoding a digital video stream into an analog video signal.

Independent Attributes

Table 1014

Name	Description
`encodingType`	The current encoding type used for encoding the analog video signal.
`supportedEncodingTypes`	Vector that contains all supported encoding types. The vector is constructed by logical OR-ing the supported encoding types.
`colorSystem`	The current color system that is used for encoding the analog video signal.
`supportedColorSystems`	Vector that contains all supported color systems. The vector is constructed by logical OR-ing the supported color systems.

Dependent Attributes

None.

Invariants

— `(supportedEncodingTypes & encodingType) != 0`

— `(supportedColorSystems & colorSystem) != 0`

Instantiation

See subclause 8.2.2.3.3.

Streaming Behavior

The `AnaVenc` has a digital video stream as input. This video stream is encoded into an analog video signal using the selected encoding type and the selected color system.

Active Behavior

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Execution Constraints
None.

8.2.2.4.2 AnaVencClient

Signature
```
role AnaVencClient {}
```

Qualifiers
— actor

Description
An instance of the role `AnaVencClient` represents a client of the analog video encoding functionality as provided by an Analog Video Encoder.

8.2.2.5 Interfaces

8.2.2.5.1 uhIAnaVenc

Qualifiers
— None.

Description
This interface provides control over the Analog Video Encoder. It provides functions to set the encoding type and the color system.

Interface ID
```
uuid( F14EACA3-C9AB-4b00-BE4A-02A20C599416 )
```

Remarks
In the following function specifications, the name `anaVenc will b`e used to refer to the provider of this interface; see subclause 8.2.2.3.3.1

8.2.2.5.1.1 SetEncodingType

Signature
```
uhErrorCode_t SetEncodingType (
    [in] uhAnaVenc_EncodingType_t encodingType );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the encoding type that is used for encoding the analog output signal.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1015

Name	Description
`encodingType`	The encoding type.

Return Values
Standard.

Precondition
— `(anaVenc.supportedEncodingTypes &` encodingType`) != 0`

Action
— modify `anaVenc.encodingType`

Postcondition
— `anaVenc.encodingType ==` *encodingType*

8.2.2.5.1.2 GetEncodingType

Signature
```
uhErrorCode_t GetEncodingType (
    [out] puhAnaVenc_EncodingType_t pEncodingType );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current encoding type that is used for encoding the analog output signal.

Parameters

Table 1016

Name	Description
`pEncodingType`	Output parameter returning the current encoding type.

Return Values
Standard.

Precondition
`True`

Action
— None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `*pEncodingType == anaVenc.encodingType`

8.2.2.5.1.3 SetColorSystem

Signature
```
uhErrorCode_t SetColorSystem (
    [in]  uhAnaVenc_ColorSystem_t colorSystem );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the color system that is used for the encoding of the analog video signal.

Parameters

Table 1017

Name	Description
`colorSystem`	The color system used for the encoding of the analog video signal.

Return Values
Standard.

Precondition
`(anaVenc.supportedColorSystems & colorSystem) != 0`

Action
— modify `anaVenc.colorSystem`

Postcondition
— `anaVenc.colorSystem ==` *colorSystem*

8.2.2.5.1.4 GetColorSystem

Signature
```
uhErrorCode_t GetColorSystem (
    [out] puhAnaVenc_ColorSystem_t pColorSystem );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the color system that is used for encoding the analog video signal.

© ISO/IEC 2007 – All rights reserved
--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1018

Name	Description
pColorSystem	Output parameter returning the current color system that is used for encoding the analog video signal.

Return Values
Standard.

Precondition
True

Action
— None.

Postcondition
— *pColorSystem == anaVenc.colorSystem

8.2.2.5.1.5 GetSuppEncodingType

Signature

```
uhErrorCode_t GetSuppEncodingTypes (
    [out]  puhAnaVenc_EncodingTypeSet_t pSupportedEncodingTypes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns a vector that contains the encoding types that are supported by the Analog Video Encoder. The vector is constructed by logical OR-ing the supported encoding types.

Parameters

Table 1019

Name	Description
pSupportedEncodingTypes	Output parameter returning a vector with supported encoding types.

Return Values
Standard.

Precondition
True

Action
— None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `*pSupportedEncodingTypes == anaVenc.supportedEncodingTypes`

8.2.2.5.1.6 GetSuppColorSystems

Signature
```
uhErrorCode_t GetSuppColorSystems (
    [out]  puhAnaVenc_ColorSystemSet_t pSupportedColorSystems );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns a vector that contains the color systems that are supported by the Analog Video Encoder. The vector is constructed by logical OR-ing the supported color systems.

Parameters

Table 1020

Name	Description
`pSupportedColorSystems`	Output parameter returning a vector with supported color systems.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*pSupportedColorSystems == anaVenc.supportedColorSystems`

8.2.3 UhlAnaVencrypt

8.2.3.1 Concepts

8.2.3.1.1 Introduction

This document defines the Analog Video Encryption logical component (LC). The Analog Video Encryptor gives control over an anti-taping feature.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.3.1.2 Analog Video Encryption

The Analog Video Encryptor modifies an analog video signal in such a way that copying the video while maintaining the original quality is not possible anymore. The applied copy-protection is transparent when viewing the original program. But when a copy is made, the quality of the video degrades to such an extent that it loses its value. The copy protection feature will be called anti-taping in the rest of this document.

8.2.3.1.3 Macrovision

The Macrovision Copy Protection process provides functionality for the anti-taping feature. The process consists of two primary aspects: the “AGC” process and the “Colorstripe” process. Each of these processes creates modifications to the standard analog video signal designed to exploit fundamental differences in the way display products and VCR products use the various component parts of an analog video signal.

As its name implies, the AGC process disturbs the luminance automatic gain control (AGC) circuit in the VCR. The encoded video signal appears to the AGC circuit as an abnormally high amplitude signal, which the AGC circuit corrects for, resulting in dark distorted pictures. The AGC process consists of luminance signal components only.

The colorstripe process creates a modified color burst on small groups of selected lines throughout the picture. The VCR uses the color burst to perform functions that a display device does not have to contend with and therefore has problems taping it correctly. The colorstripe encoded color burst waveform, as its name implies, results in color stripes appearing across the picture of the copy while leaving the directly displayed picture quality unaffected. The colorstripe process consists of chrominance signal components only.

For more detailed information about the Macrovision anti-taping process, see [11].

8.2.3.2 Types & Constants

8.2.3.2.1 Public Types & Constants

8.2.3.2.1.1 Error Codes

None.

8.2.3.2.1.2 uhAnaVencrypt_Feature_t

Signature

```
typedef enum _uhAnaVencrypt_Feature_t {
    uhAnaVencrypt_SyncReductionInsideVbi   = 0x00000001,
    uhAnaVencrypt_SyncReductionOutsideVbi  = 0x00000002,
    uhAnaVencrypt_AgcCycling               = 0x00000004,
    uhAnaVencrypt_ColorStripe              = 0x00000008,
    uhAnaVencrypt_BackPorchPulses          = 0x00000010,
    uhAnaVencrypt_VbiPulses                = 0x00000020
} uhAnaVencrypt_Feature_t, *puhAnaVencrypt_Feature_t;
```

Qualifiers

— enum-element

Description

Describes the different anti-taping features.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1021

Name	Description
uhAnaVencrypt_SyncReductionInsideVbi	Sync amplitude reduction feature for lines inside VBI. When set, the feature is enabled. Otherwise, the feature is disabled.
uhAnaVencrypt_SyncReductionOutsideVbi	Sync amplitude reduction feature for lines outside VBI. When set, the feature is enabled. Otherwise the feature is disabled.
uhAnaVencrypt_AgcCycling	AGC pulse amplitude feature. When set, the amplitude is cycling. Otherwise, the amplitude is static.
uhAnaVencrypt_ColorStripe	Colorstripe feature. When set, the feature is enabled. Otherwise, the feature is disabled.
uhAnaVencrypt_BackPorchPulses	End of field back porch pulses feature. When set, the feature is enabled. Otherwise, the feature is disabled.
uhAnaVencrypt_VbiPulses	PS and AGC pulses feature. When set, the feature is enabled. Otherwise, the feature is disabled.

8.2.3.2.1.3 uhAnaVencrypt_FeatureSet_t

Signature

```
typedef Int32 uhAnaVencrypt_FeatureSet_t, *puhAnaVencrypt_FeatureSet_t;
```

Qualifiers

— enum-set

Description

Values of this type identify sets of anti-taping features which are typically constructed by logical OR-ing values of type `uhAnaVencrypt_Feature_t`.

8.2.3.2.1.4 uhAnaVencrypt_ColorStripe_t

Signature

```
typedef struct _uhAnaVencrypt_ColorStripe_t {
    UInt8 firstLineFirstStripeField1;
    UInt8 spacingFirstToSecondStripeField1;
    UInt8 firstLineFirstStripeField2;
    UInt8 spacingFirstToSecondStripeField2;
    UInt8 spacingFromSecondStripe;
    UInt8 nrStripesPerField;
    UInt8 nrLinesPerStripe;
} uhAnaVencrypt_ColorStripe_t, *puhAnaVencrypt_ColorStripe_t;
```

Qualifiers

None.

Description

Defines the attributes used for selecting the line numbers for which the waveform is modified by the Macrovision Copy Protection Process. These lines are so called colorstripe lines.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Fields

Table 1022

Name	Description
firstLineFirstStripeField1	First line in first colorstripe. Field 1 / Even field.
spacingFirstToSecondStripeField1	Colorstripe spacing, first to second stripe. Field 1 / Even field.
firstLineFirstStripeField2	First line in first colorstripe. Field 2 / Odd field.
spacingFirstToSecondStripeField2	Colorstripe spacing, first to second stripe. Field 2 / Odd field.
spacingFromSecondStripe	Colorstripe spacing, from second stripe in each field.
nrStripesPerField	Number of colorstripes per field.
nrLinesPerStripe	Number of lines per colorstripe.

Constraints

— Values for the fields of this type must comply with the Macrovision specification [11].

8.2.3.2.1.5 uhAnaVencrypt_BurstWaveform_t

Signature

```
typedef struct _uhAnaVencrypt_BurstWaveform_t {
    Bool  advanceBurstStart;
    UInt8 startBurstToFirstPhaseSwitch;
    UInt8  firstToSecondPhaseSwitch;
    UInt8  secondPhaseSwitchToEndBurst;
    UInt8  subcarrierPhaseZones;
    UInt16 stripeLinePhases;
} uhAnaVencrypt_BurstWaveform_t, *puhAnaVencrypt_BurstWaveform_t;
```

Qualifiers

None.

Description

Defines the attributes that characterize the burst waveform used on those lines selected as the colorstripe lines.

Fields

Table 1023

Name	Description
advanceBurstStart	Advanced burst start On / Off (True / False).
startBurstToFirstPhaseSwitch	Duration of start of burst to first phase switch point (zone 1).
firstToSecondPhaseSwitch	Duration of first to second phase switch point (zone 2).
secondPhaseSwitchToEndBurst	Duration of second phase switch to end of burst (zone 3).
subcarrierPhaseZones	Subcarrier phase zones (zone 1 to 3).
stripeLinePhases	Colorstripe lines phases (line phase 1 to 5).

Constraints

— Values for the fields of this type must comply with the Macrovision specification [11].

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.3.2.1.6 uhAnaVencrypt_VbiPulse_t

Signature

```
typedef struct _uhAnaVencrypt_VbiPulse_t  {
    UInt8   durationPseudoSyncPulseA;
    UInt8   durationPseudoSyncPulseB;
    UInt8   locationFirstPseudoSyncPulseA;
    UInt8   locationFirstPseudoSyncPulseB;
    UInt8   spacingPseudoSyncPulseA;
    UInt8   spacingPseudoSyncPulseB;
    UInt16  lineNumberSelection;
    UInt16  lineNumberFormatSelection;
    UInt8   pulseConfigurationA;
    UInt8   pulseConfigurationB;
    UInt8   nrBackPorchPulsesPriorVsync;
    UInt8   nrBackPorchPulsesAfterVsync;
} uhAnaVencrypt_VbiPulse_t, *puhAnaVencrypt_VbiPulse_t;
```

Qualifiers
None.

Description
Defines the attributes used in the AGC process. The AGC process consists primarily of creating a number of pulse pairs in and around the VBI data.

Fields

Table 1024

Name	Description
durationPseudoSyncPulseA	PS pulse duration, format A.
durationPseudoSyncPulseB	PS pulse duration, format B.
locationFirstPseudoSyncPulseA	First PS pulse location, format A.
locationFirstPseudoSyncPulseB	First PS pulse location, format B.
spacingPseudoSyncPulseA	PS pulse spacing, format A.
spacingPseudoSyncPulseB	PS pulse spacing, format B.
lineNumberSelection	Line number selections with PS/AGC pulses.
lineNumberFormatSelection	Line numbers with A/B format PS/ AGC pulses.
pulseConfigurationA	PS/AGC pulse line configuration, format A.
pulseConfigurationB	PS/AGC pulse line configuration, format B.
nrBackPorchPulsesPriorVsync	Number of back porch pulses before to V-sync.
nrBackPorchPulsesAfterVsync	Number of back porch pulses after V-sync.

Constraints
— Values for the fields of this type must comply with the Macrovision specification [11].

8.2.3.2.2 Model Types & Constants

None.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.3.3 Logical Component

8.2.3.3.1 Interface-Role Model

Figure 109 depicts the interface-role model of the Analog Video Encryption LC. it shows the roles and interfaces the Analog Video Encryption LC and its client can have.

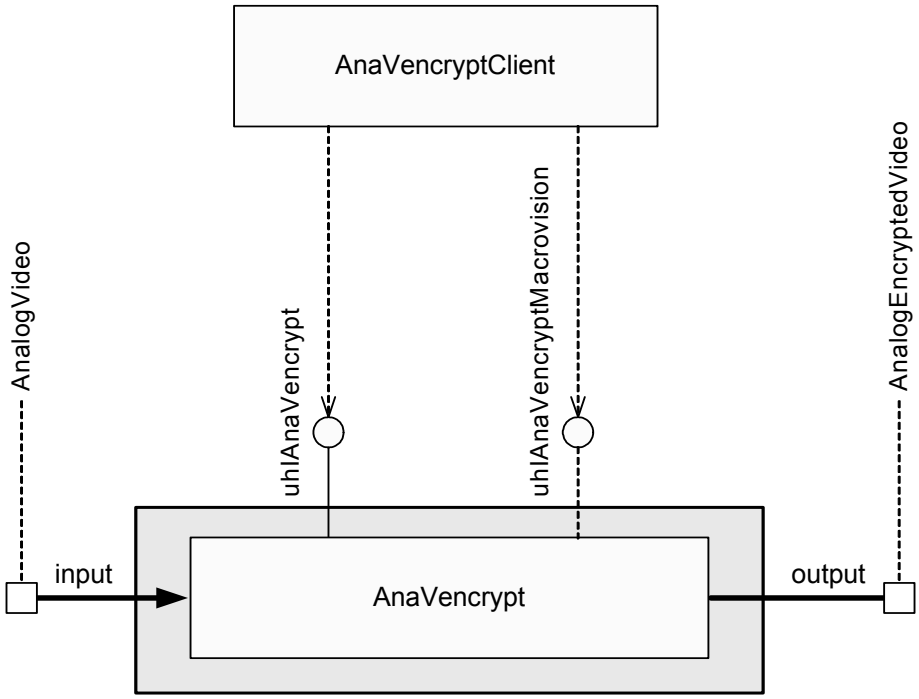


Figure 109 — Interface-Role Model of the Analog Video Encryptor

The Analog Video Encryption LC (grey box) consists of the `AnaVencrypt` role. This role provides two interfaces. The `uhIAnaVencrypt` interface controls whether the anti-taping feature is enabled or disabled. The `uhIAnaVencryptMacrovision` interface controls the anti-taping feature as developed by Macrovision. The control interfaces are meant for usage by the client `AnaVencryptClient`.

8.2.3.3.2 Diversity

8.2.3.3.2.1 Provided Interfaces

Table 1025

Role	Interface	Presence
`anaVencrypt`	`uhIAnaVencrypt`	mandatory
`anaVencrypt`	`uhIAnaVencryptMacrovision`	optional

8.2.3.3.2.2 Configurable Items

None.

8.2.3.3.2.3 Constraints

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.3.3.3 Instantiation

8.2.3.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1026

Type	Object	Multiplicity
`AnaVencrypt`	`anaVencrypt`	1

8.2.3.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

- `anaVencrypt.enabled == False`
- `anaVencrypt.rgbOutputProtectionEnabled == False`
- `anaVencrypt.featureSetup == 0`

The following attributes are structs with undefined values:

- `anaVencrypt.colorStripeSetup`
- `anaVencrypt.burstWaveformSetup`
- `anaVencrypt.vbiPulseSetup`

8.2.3.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.2.3.4 Roles

8.2.3.4.1 AnaVencrypt

Signature

```
role AnaVencrypt {
    Bool  enabled;
    Bool  rgbOutputProtectionEnabled;
    uhAnaVencrypt_FeatureSet_t    featureSetup;
    uhAnaVencrypt_ColorStripe_t   colorStripeSetup;
    uhAnaVencrypt_BurstWaveform_t burstWaveformSetup;
    uhAnaVencrypt_VbiPulse_t      vbiPulseSetup;
}
```

Qualifiers

- root

Description

The `AnaVencrypt` provides functionality to enable, disable and configure the anti-taping feature.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 1027

Name	Description
`enabled`	Indicates whether the anti-taping feature is enabled (True) or not (False).
`rgbOutputProtectionEnabled`	Enables or disables copy protection of waveforms for RGB output.
`featureSetup`	Current set of selected anti-taping features.
`colorStripeSetup`	Attributes for selecting the colorstripe lines.
`burstWaveformSetup`	Attributes for setting the colorstripe burst waveform.
`vbiPulseSetup`	Attributes for configuring the AGC process.

Dependent Attributes
None.

Invariants

None.

Instantiation
See subclause 8.2.3.3.3.

Streaming Behavior
The `AnaVencrypt` has an analog video signal as input. When the anti-taping feature is enabled, the analog video signal is modified in such a way that the signal cannot be taped with the original video quality. The modified analog signal is passed to the output. When the anti-taping feature is disabled, the analog video signal at the input is passed to the output without modification.

Active Behavior
None.

8.2.3.4.2 AnaVencryptClient

Signature
```
role AnaVencryptClient {}
```

Qualifiers
— actor

Description
An instance of the role `AnaVencryptClient` represents a client of the anti-taping functionality as provided by an Analog Video Encryptor.

8.2.3.5 Interfaces

8.2.3.5.1 uhIAnaVencrypt

Qualifiers
— None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface provides functionality to enable or disable anti-taping functionality.

Interface ID
`uuid( 3832FFB9-FF41-49e9-81F5-E73AB09C2292 )`

Remarks
In the following function specifications, the name `anaVencrypt` will be used to refer to the provider of this interface; see subclause 8.2.3.3.3.1.

8.2.3.5.1.1 Enable

Signature
```
uhErrorCode_t Enable (
    [in]  Bool enabled );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables the anti-taping feature.

Parameters

Table 1028

Name	Description
`enabled`	Boolean to enable (True) or disable (False) the anti-taping feature.

Return Values
Standard

Precondition
`True`

Action
— Modify `anaVencrypt.enabled`

Postcondition
— `anaVencrypt.enabled == enabled`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.3.5.1.2 GetEnabled

Signature
```
uhErrorCode_t GetEnabled (
    [out]  pBool pEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the anti-taping feature is enabled or disabled.

Parameters

Table 1029

Name	Description
`pEnabled`	Output parameter returning whether the anti-taping feature is enabled (`True`) or disabled (`False`).

Return Values
Standard

Precondition
`True`

Action
— None.

Postcondition
— `*`pEnabled `== anaVencrypt.enabled`

8.2.3.5.2 uhIAnaVencryptMacrovision

Qualifiers
None.

Description
This interface provides functions to set Macrovision data to control the anti-taping feature. For more information about Macrovision and the meaning of the data, see [11].

Interface ID
`uuid( 6980D3D0-D741-4877-AE23-815D954DCBB9 )`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks
In the following function specifications, the name `anaVencrypt` will be used to refer to the provider of this interface; see subclause 8.2.3.3.3.1

8.2.3.5.2.1 SetFeatures

Signature
```
uhErrorCode_t SetFeatures (
    [in]  uhAnaVencrypt_FeatureSet_t featureSetup );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the Macrovision anti-taping features.

Parameters

Table 1030

Name	Description
`featureSetup`	Vector with the selected anti-taping features.

Return Values
Standard

Precondition
`True`

Action
— Modify `anaVencrypt.featureSetup`

Postcondition
— `anaVencrypt.featureSetup ==` *featureSetup*

8.2.3.5.2.2 SetColorStripe

Signature
```
uhErrorCode_t SetColorStripe (
    [in]  uhAnaVencrypt_ColorStripe_t  colorStripeSetup );
```

Qualifiers
— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Sets the attributes used for selecting the line numbers for which the waveform is modified by the Macrovision Copy Protection process. These lines are so called colorstripe lines.

Parameters

Table 1031

Name	Description
`colorStripeSetup`	Parameter for selecting the colorstripe lines.

Return Values
Standard

Precondition
`True`

Action
— Modify `anaVencrypt.colorStripeSetup`

Postcondition
— `anaVencrypt.colorStripeSetup ==` colorStripeSetup

8.2.3.5.2.3 SetBurstWaveform

Signature
```
uhErrorCode_t SetBurstWaveform (
    [in]  uhAnaVencrypt_BurstWaveform_t  burstWaveformSetup );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the attributes that characterize the burst waveform used on those lines selected as the colorstripe lines.

Parameters

Table 1032

Name	Description
`burstWaveformSetup`	Parameter to set the colorstripe burst waveform.

Return Values
Standard

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `anaVencrypt.burstWaveformSetup`

Postcondition
— `anaVencrypt.burstWaveformSetup ==` *burstWaveformSetup*

8.2.3.5.2.4 SetVbiPulse

Signature
```
uhErrorCode_t SetVbiPulse (
    [in] uhAnaVencrypt_VbiPulse_t  vbiPulseSetup );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the attributes used in the AGC process. The AGC process consists primarily of creating a number of pulse pairs in and around the VBI data.

Parameters

Table 1033

Name	Description
`vbiPulseSetup`	Parameter for the AGC process.

Return Values
Standard

Precondition
`True`

Action
— Modify `anaVencrypt.vbiPulseSetup`

Postcondition
— `anaVencrypt.vbiPulseSetup ==` vbiPulseSetup

8.2.3.5.2.5 EnableRgbOutputProtection

Signature
```
uhErrorCode_t EnableRgbOutputProtection (
    [in]  Bool  rgbOutputProtectionEnabled );
```

Qualifiers
— synchronous

— single-threaded

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Enables or disables the RGB output copy protection.

Parameters

Table 1034

Name	Description
`rgbOutputProtectionEnabled`	Parameter to enable (`True`) or disable (`False`) RGB output copy protection.

Return Values
Standard

Precondition
`True`

Action
— Modify `anaVencrypt.rgbOutputProtectionEnabled`

Postcondition
— `anaVencrypt.rgbOutputProtectionEnabled==` *rgbOutputProtectionEnabled*

8.2.4 UhlAntiAging

8.2.4.1 Concepts

8.2.4.1.1 Introduction

Unlike conventional CRT, other display types, like plasma and LCD, may have the peculiarity to suffer from aging effects if a static picture (or a broadcaster's logo) remains active on the display for longer periods. To reduce these "burn-in" effects in this kind of matrix panels (in fact, aging of display phosphors for plasma and image sticking for LCD), a counter measure is provided, called "Anti-Aging". In short, this feature slowly shifts the picture in horizontal and vertical direction to spread the aging effects over a bigger area. The small shift needs to be invisible for a customer.

Figure 110 (upper half) shows how the entire picture is shifted across the screen (the red dot is the initial “centered” position). The shifting area in this case is 4 pixels (right and left) and 3 lines (up and down). The size of the shifting area is limited by the amount of overscan that is available (overscan is the part of the input image that is not displayed).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

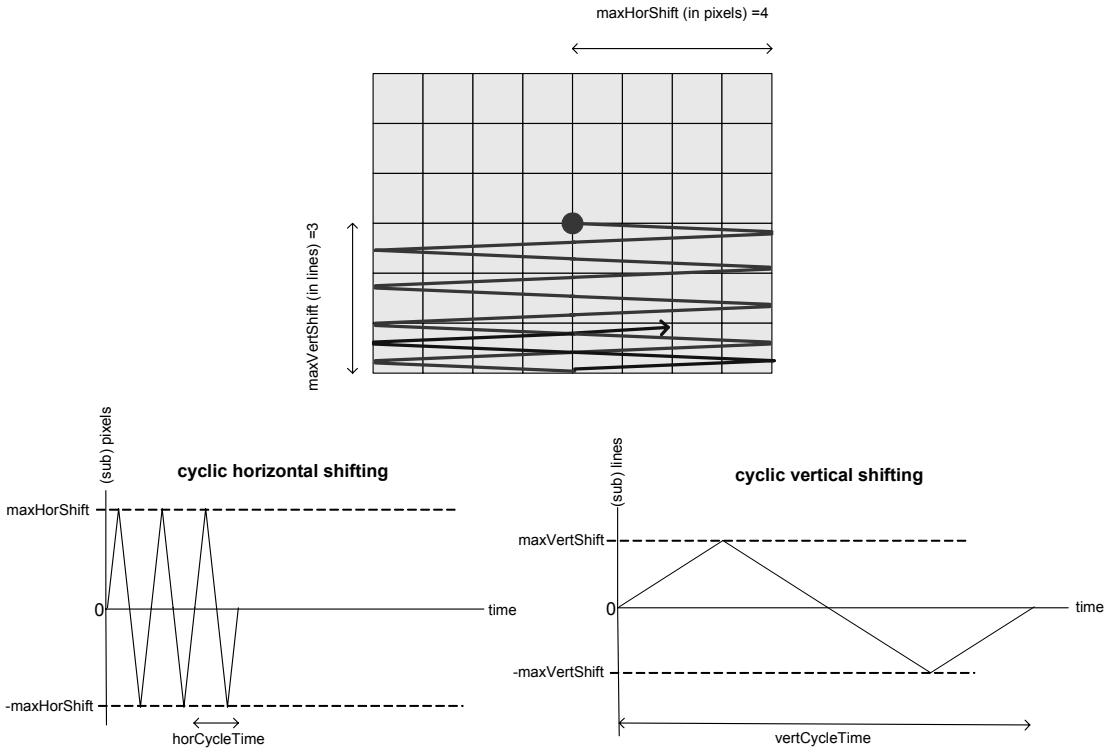


Figure 110 — The anti-aging process: a combination of horizontal and vertical cyclic shifting.

If not enough overscan is available, e.g. anti-aging wants to shift 2.7 pixels to the right, but there are only 2 pixels overscan to the right, then the platform shifts the picture only 2 pixels (the actual shift is clipped). The client should actually take care that enough overscan is available or turn anti-aging off if that is not the case.

The movement of the picture can be decomposed into a cyclic horizontal shift process and a cyclic vertical shift process (lower half of Figure 110). The respective speeds of these processes (`horCycleTime` and `vertCycleTime`) should be set in such a way that a large part of the shifting area is covered, so that the aging effect is spread as much as possible. In the example, horizontal shifting is a lot faster than vertical shifting to achieve this.

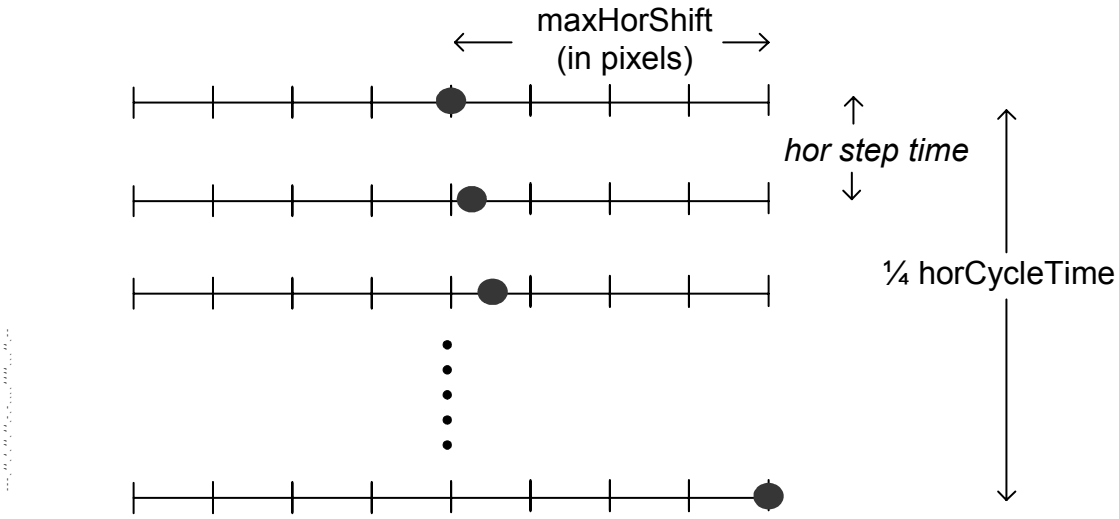
Because the slower of the two shift processes typically takes a few hours to complete a full cycle, it is best to store the last shift position in non-volatile memory when the television set is switched off, so that the anti-aging process can continue from this position when the set is switched on again. This will guarantee a more homogeneous spread of the picture shift over long periods.

More important for spreading the aging effects and for keeping the anti-aging invisible, is the (sub-pixel/line) resolution of the shifts. The resolution determines how smooth the shifting is. It can be expressed as the number of single shift steps that can be used to cover the distance of a single pixel or line (`horStepSize` or `vertStepSize`). So, the actual distance covered by a single shift step is 1/horStepSize pixel or 1/vertStepSize line.

Figure 111 shows how the time inbetween two horizontal shift steps (hor step time) can be derived from `horCycleTime`, `maxHorShift`, and `horStepSize`. A similar derivation can be made for the time between two vertical shift steps.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



$$hor\ step\ time = \frac{horCycleTime}{4 * maxHorShift * horStepSize}$$

Figure 111 — Deriving hor step time from horCycleTime, maxHorShift, and horStepSize.

8.2.4.2 Types & Constants

8.2.4.2.1 Public Types & Constants

8.2.4.2.1.1 uhAntiAging_Mode_t

Signature

```
typedef enum _uhAntiAging_Mode_t {
    uhAntiAging_Off         = 0x00000001,
    uhAntiAging_Horizontal  = 0x00000002,
    uhAntiAging_Vertical    = 0x00000004,
    uhAntiAging_Both        = 0x00000008
} uhAntiAging_Mode_t, *puhAntiAging_Mode_t;
```

Qualifiers

— enum-element

Description

Values of this type represent modes of operation of the anti-aging algorithm.

Values

Table 1035

Name	Description
uhAntiAging_Off	No anti-aging.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
`uhAntiAging_Horizontal`	Only horizontal anti-aging.
`uhAntiAging_Vertical`	Only vertical anti-aging.
`uhAntiAging_Both`	Both horizontal and vertical anti-aging.

8.2.4.2.1.2 uhAntiAging_Modes_t

Signature
```
typedef Int32 uhAntiAging_Modes_t, *puhAntiAging_Modes_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of anti-aging modes which are typically constructed by logical OR-ing of values of type uh`AntiAging_Mode_t`.

8.2.4.3 Logical Component

8.2.4.3.1 Interface-Role Model

Figure 112 depicts the interface-role model of the AA (the grey box in Figure 112). It shows the control interface and the roles involved in the AA.

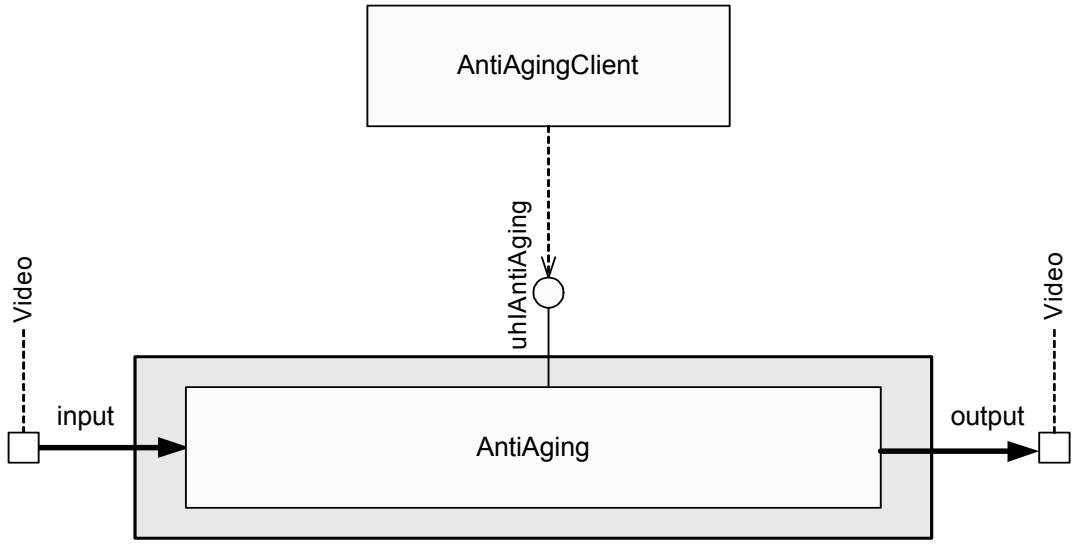


Figure 112 — Interface-Role Model

An AA contains the role AntiAging which provides the control interface `uhIAntiAging`. A client (represented by the AntiAgingClient role) can use this control interface.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.4.3.2 Diversity

8.2.4.3.2.1 Provided Interfaces

Table 1036

Role	Interface	Presence
AntiAging	uhIAntiAging	mandatory

8.2.4.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of an AA. For a description of the attributes, see AntiAging, subclause 8.2.4.4.1.

Table 1037

Role	Attribute
AntiAging	horCycleTime
	vertCycleTime
	maxHorShift
	maxVertShift
	suppModes
	horStepSize
	vertStepSize

8.2.4.3.2.3 Constraints

— horCycleTime > 0
— vertCycleTime > 0

Additionally, the hardware will pose certain restrictions on the values that can be given to the configurable items listed above.

8.2.4.3.3 Instantiation

8.2.4.3.3.1 Objects Created

The following object is created when the logical component is instantiated:

Table 1038

Type	Object	Multiplicity
AntiAging	aag	1

8.2.4.3.3.2 Initial State

— aag.mode == uhAntiAging_Off
— aag.curHorPos == 0

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— aag.curVertPos == 0

8.2.4.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.2.4.4 Roles

8.2.4.4.1 AntiAging

Signature

```
role AntiAging {
    uhAntiAging_Mode_t          mode;
    const UInt32                maxHorShift;
    const UInt32                maxVertShift;
    const UInt32                horCycleTime;
    const UInt32                vertCycleTime;
    UInt32                      curHorPos;
    UInt32                      curVertPos;
    const uhAntiAging_Modes_t   suppModes;
    const UInt32                horStepSize;
    const UInt32                vertStepSize;
}
```

Qualifiers
Root.

Description
The AntiAging role applies the anti-aging algorithm to a video stream.

Independent Attributes

Table 1039

Name	Description
mode	The current anti-aging mode.
maxHorShift	The maximum horizontal picture shift, to the left and to the right (in ‰). The maximum allowed shift (in pixels) of the input (to the right or left) then is: $\frac{\text{maxHorShift} \cdot \text{width of displayed area}}{1000}$ See Figure 113.
maxVertShift	The maximum vertical picture shift, up and down (in ‰). The maximum allowed shift (in lines) of the input (up or down) then is: $\frac{\text{maxVertShift} \cdot \text{height of displayed area}}{1000}$ See Figure 113.
horCycleTime	The horizontal cycle time (in milliseconds).
vertCycleTime	The vertical cycle time (in milliseconds).
curHorPos	The current horizontal position of the anti-aging algorithm (in milliseconds).
curVertPos	The current vertical position of the anti-aging algorithm (in milliseconds).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
`suppModes`	The supported anti-aging modes.
`horStepSize`	The horizontal shift resolution at the output (steps of $\frac{1}{\text{horStepSize}}$ pixel).
`vertStepSize`	The vertical shift resolution at the output (steps of $\frac{1}{\text{vertStepSize}}$ line).

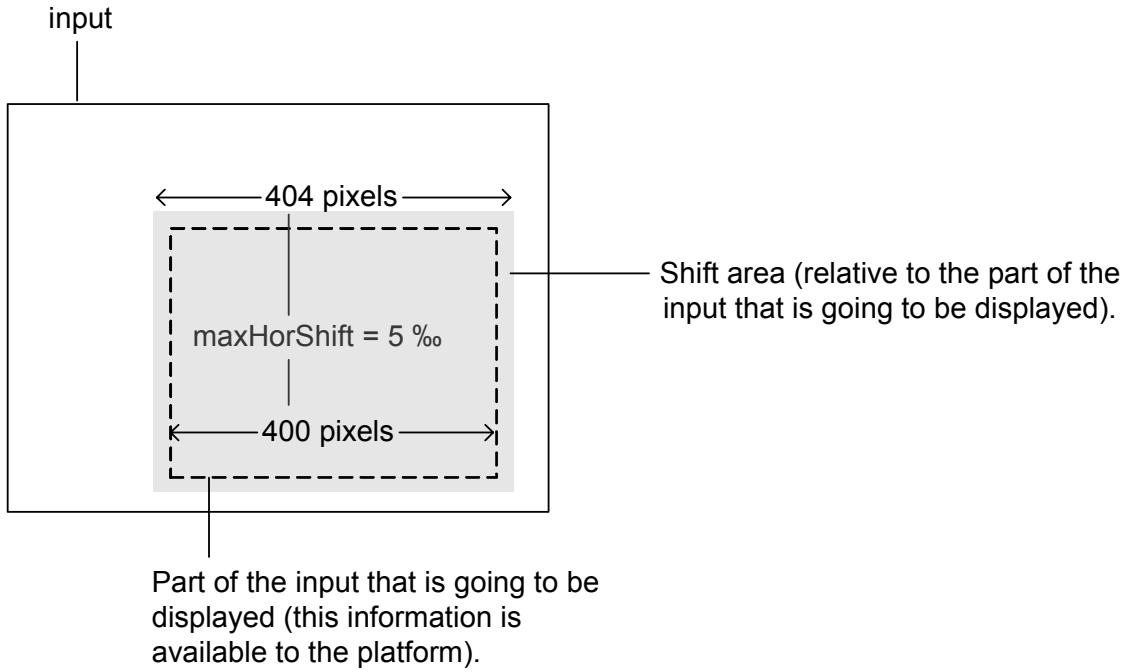


Figure 113 — Specification of the size of the shift area.

Dependent Attributes

None.

Invariants

— `mode` in `suppModes`
— `0 <= curHorPos < horCycleTime`
— `0 <= curVertPos < vertCycleTime`

Instantiation
The `AntiAging` role is always instantiated as part of an AA.

Streaming Behavior
The AntiAging role applies a horizontal and/or a vertical shift of subpixel resolution to the incoming video and streams the result to the output of the AA. See Figure 114 for an example. Note that the output stream of the AA is conceptual; there does not need to be a software or hardware component that actually outputs such images. The anti-aging algorithm varies the shifts in time. The client has some control over this algorithm via the `uhIAntiAging` interface.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

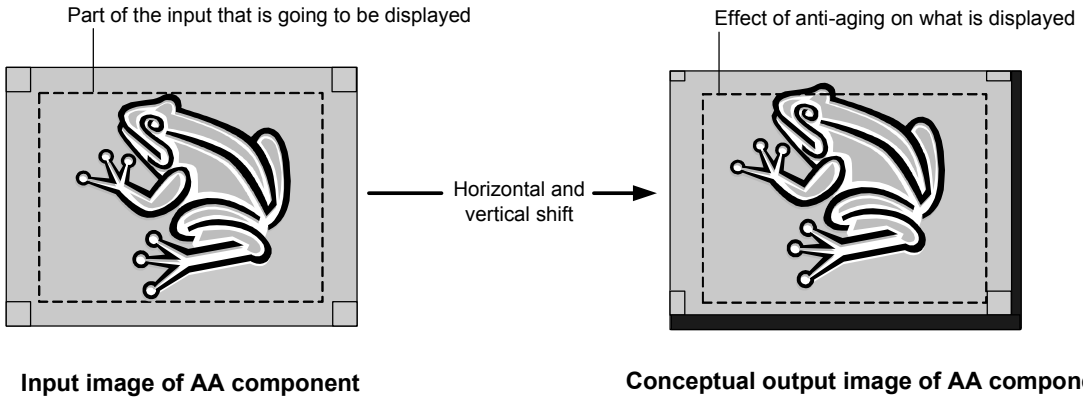


Figure 114 — Processing of a single image by the Anti-Aging component.

Active Behavior
None.

8.2.4.4.2 AntiAgingClient

Signature
`role AntiAgingClient {}`

Qualifiers
— actor

Description
A `AntiAgingClient` represents a user of the `uhIAntiAging` interface. No constraints apply to the behavior of the `AntiAgingClient` other than those implied by Figure 112.

8.2.4.5 Interfaces

8.2.4.5.1 uhIAntiAging

Qualifiers
None.

Description
This interface of the AntiAging role allows control over the anti-aging algorithm.

Interface ID
`uuid( 658A2801-38AF-48e9-8FBD-D42BD76F68D7 )`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.4.5.1.1 GetSuppModes

Signature
```
uhErrorCode_t GetSuppModes ( [out] puhAntiAging_Modes_t    pSuppModes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the supported anti-aging modes.

Parameters

Table 1040

Name	Description
`pSuppModes`	Output parameter returning the supported anti-aging modes.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*`pSuppModes `== aag.suppModes`

8.2.4.5.1.2 GetMode

Signature
```
uhErrorCode_t GetMode ( [out] puhAntiAging_Mode_t    pMode );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current anti-aging mode.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1041

Name	Description
pMode	Output parameter returning the current anti-aging mode.

Return Values
Standard.

Precondition
True

Action
— None.

Postcondition
— **pMode* == aag.mode

8.2.4.5.1.3 SetMode

Signature
```
uhErrorCode_t SetMode ( [in] uhAntiAging_Mode_t mode );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the mode of the anti-aging algorithm. If anti-aging is turned off, shifting stops, but the current horizontal and vertical offsets of the image are maintained (also, if anti-aging in one of the two directions is turned off, the offset in that direction is maintained).

Parameters

Table 1042

Name	Description
mode	The new mode of the anti-aging algorithm.

Return Values
Standard

Precondition
— mode in aag.suppModes

Action
— Modify aag.mode

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `aag.mode == mode`

8.2.4.5.1.4 GetCycleTime

Signature
```
uhErrorCode_t GetCycleTime (
    [out] pUInt32              pHorCycleTime,
    [out] pUInt32              pVertCycleTime );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the horizontal and vertical cycle times of the anti-aging algorithm.

Parameters

Table 1043

Name	Description
`pHorCycleTime`	Output parameter returning the horizontal cycle time of the anti-aging algorithm (in milliseconds).
`pVertCycleTime`	Output parameter returning the vertical cycle time of the anti-aging algorithm (in milliseconds).

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*`*pHorCycleTime* `== aag.horCycleTime`
— `*`pVertCycleTime `== aag.vertCycleTime`

8.2.4.5.1.5 GetCurPos

Signature
```
uhErrorCode_t GetCurPos (
    [out] pUInt32           pCurHorPos,
    [out] pUInt32           pCurVertPos );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Returns the current position of the anti-aging algorithm. The position contains information on the current horizontal and vertical shift of the picture and horizontal and vertical shift directions of the algorithm. See `SetCurPos` for more detailed information on “positions”. Note that if movement in one or two directions is switched off, the algorithm maintains its current position in the direction(s) in which the movement is switched off..

Parameters

Table 1044

Name	Description
`pCurHorPos`	Output parameter returning the current horizontal position of the anti-aging algorithm (in milliseconds).
`pCurVertPos`	Output parameter returning the current vertical position of the anti-aging algorithm (in milliseconds).

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*`*pCurHorPos* `== aag.curHorPos`
— `*`pCurVertPos `== aag.curVertPos`

8.2.4.5.1.6 SetCurPos

Signature
```
uhErrorCode_t SetCurPos (
    [in] UInt32             horPos,
    [in] UInt32             vertPos );
```

Qualifiers
— synchronous

— single-threaded

Description
Applies a horizontal and a vertical shift to the input video and sets horizontal and vertical shift directions for the anti-aging algorithm (all independent from the current anti-aging mode). Positioning is done by specifying where the algorithm is in its horizontal and vertical cycles. Figure 115 gives an example of how a vertical position is specified. A horizontal cycle starts with shifting right and a vertical cycle starts with shifting down.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

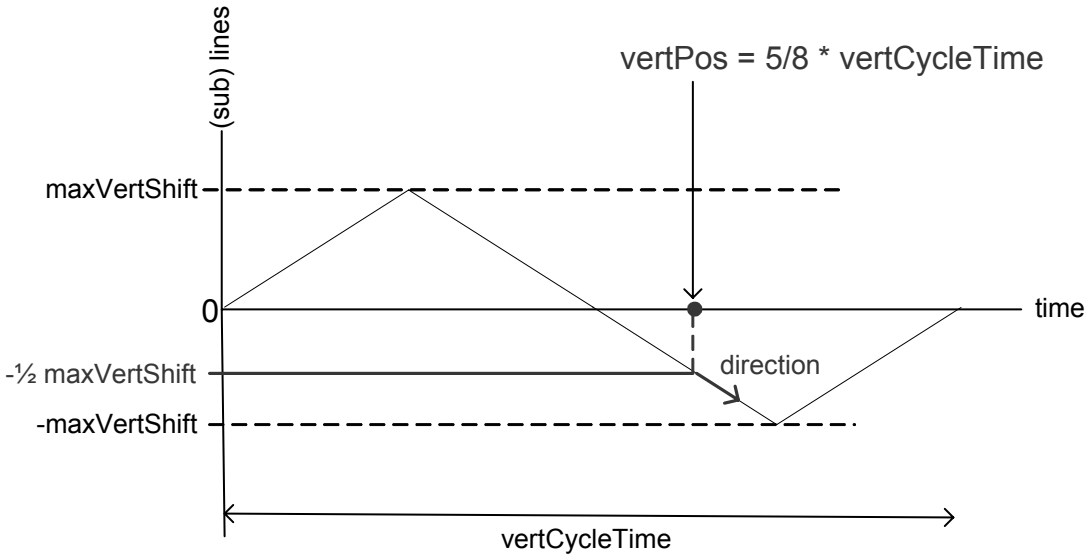


Figure 115 — Example of "positioning" the algorithm (vertically).

Parameters

Table 1045

Name	Description
`horPos`	The new horizontal position (in milliseconds).
`vertPos`	The new vertical position (in milliseconds).

Return Values
Standard.

Precondition
— `0 <=` *horPos* `< aag.horCycleTime`
— `0 <=` vertPos `< aag.vertCycleTime`
— *horPos* `== 0` or `aag.suppModes` includes at least one of the following: uh`AntiAging_Horizontal`, uh`AntiAging_Both`
— vertPos `== 0` or `aag.suppModes` includes at least one of the following: uh`AntiAging_Vertical`, uh`AntiAging_Both`

Action
— Modify `aag.curHorPos`, `aag.curVertPos`.

Postcondition
— `aag.curHorPos ==` *horPos*
— `aag.curVertPos ==` vertPos

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.5 UhlBbarDet

8.2.5.1 Concepts

8.2.5.1.1 Black Bar Detection

Pictures can be transmitted with another aspect ratio than they were recorded with. In such a case, the picture is mapped to, e.g. 4:3 or 16:9 aspect ratio and black bars are added above and below. The size of these black bars depends on the aspect ratio of the recorded picture.

Black bars can be identified by the first real (non-black) video line (`firstLine`) and the last real video line (`lastLine`). The interface described in this document abstracts from the actual black bar detection algorithm, but it does support recursive algorithms, i.e. algorithms that, when determining the position of black bars in a video frame, use information from earlier frames. Figure 116 depicts a simple example, that shows how a recursive algorithm can do a better detection than a non-recursive one. An algorithm that does not use information from previous frames may conclude that the black bars end where the white circles begin (because the background of the circles is very dark).

The interface provides a `Reset` function, that "clears the history" of the algorithm (which does not make sense for a non-recursive algorithm).

The algorithm may process fields and/or frames, but the interface is always frame based: the first and last lines that are returned by the algorithm are relative to a frame.

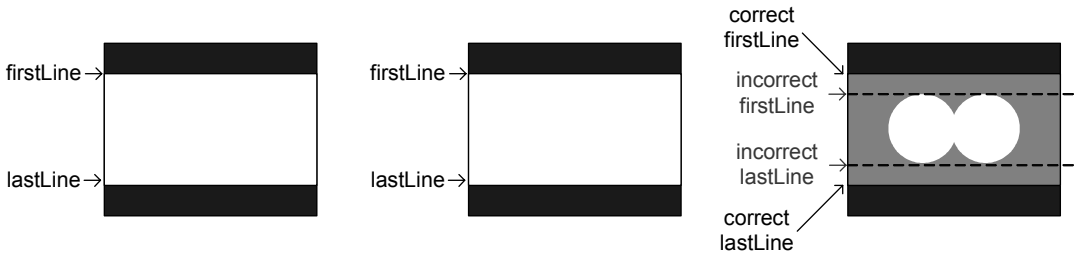


Figure 116 — Preventing a false detection by using information from previous frames.

8.2.5.2 Types & Constants

8.2.5.2.1 Public Types & Constants

8.2.5.2.1.1 uhBbarDet_ValStat_t

Signature

```
typedef enum _uhBbarDet_ValStat_t {
    uhBbarDet_ValStatInvalid = -2,
    uhBbarDet_ValStatUnknown = -1,
    uhBbarDet_ValStatKnown   =  1
} uhBbarDet_ValStat_t, *puhBbarDet_ValStat_t;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This type represents the validity status of the related property/properties.

Values

Table 1046

Name	Description
uhBbarDet_ValStatInvalid	The property/properties are temporarily not available.
uhBbarDet_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhBbarDet_ValStatKnown	The property/properties are available and determined.

8.2.5.2.1.2 uhBbarDet_Notification_t

Signature
```
typedef enum _uhBbarDet_Notification_t {

    uhBbarDet_SubscriptionChanged                  = 0x00000001,

    uhBbarDet_FirstAndLastVideoLinesChanged        = 0x00000002

} uhBbarDet_Notification_t, *puhBbarDet_Notification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIBbarDetNtf`.

Values

Table 1047

Name	Description
uhBbarDet_SubscriptionChanged	Represents `OnSubscriptionChanged`.
uhBbarDet_FirstAndLastVideoLinesChanged	Represents `OnFirstAndLastVideoLinesChanged`.

8.2.5.2.1.3 uhBbarDet_Notifications_t

Signature
```
typedef Int32 uhBbarDet_Notifications_t, *puhBbarDet_Notifications_t;
```

Qualifiers
— enum-set

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`BbarDet_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.2.5.2.1.4 uhBbarDet_AllNotifications

Signature
```
const uhBbarDet_Notifications_t uhBbarDet_AllNotifications = 0x00000003;
```

Qualifiers
— None.

Description
Defines the set of all values of type uh`BbarDet_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.2.5.2.2 Model Types & Constants

None.

8.2.5.3 Logical Component

8.2.5.3.1 Interface-Role Model

The interface role-model is depicted in Figure 117.

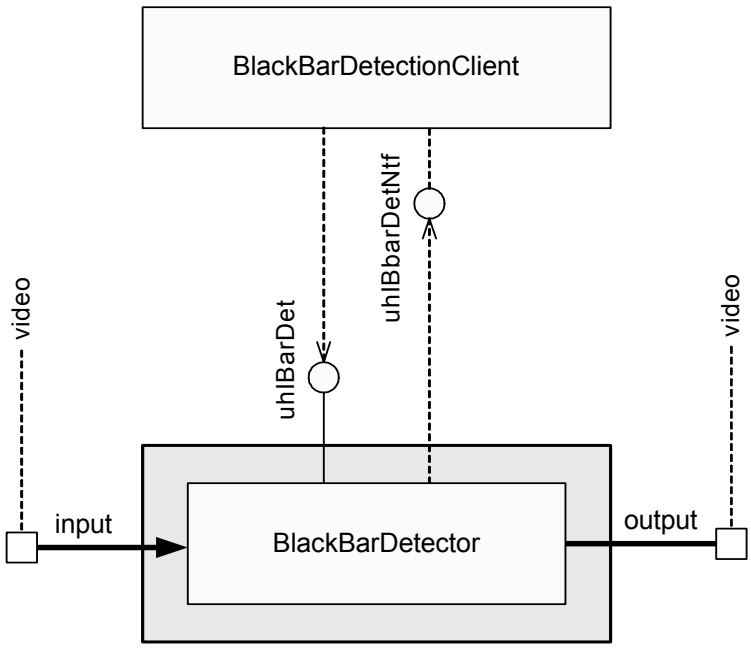


Figure 117 — Interface-Role Model of the Black Bar Detection logical component

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The BBD (grey box) consists of the `BlackBarDetector` role. This role provides an interface (`uhIBbarDet`) for controlling the black bar detection algorithm. The control interface is meant for use by the client (`BlackBarDetectionClient`). This client can optionally provide a call-back interface (`uhIBbarDetNtf`) allowing it to subscribe to notifications of events in the BBD.

8.2.5.3.2 Diversity

8.2.5.3.2.1 Provided Interfaces

Table 1048

Role	Interface	Presence
`BlackBarDetector`	`uhIBbarDet`	mandatory
`BlackBarDetectionClient`	`uhIBbarDetNtf`	optional

8.2.5.3.2.2 Configurable Items

None.

8.2.5.3.2.3 Constraints

None.

8.2.5.3.3 Instantiation

8.2.5.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1049

Type	Object	Multiplicity
`BlackBarDetector`	`bbd`	1

8.2.5.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— `bbd.vertRes` == vertical resolution of input
— `bbd.firstLine` == 0
— `bbd.lastLine` == `bbd.vertRes-1`
— `bbd.status` == `uhBbarDet_ValStatInvalid`
— The vertical resolution is frame based, so if, for instance, the input is PAL in the initial state, then `bbd.lastLine` == 575.

8.2.5.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.5.4 Roles

8.2.5.4.1 BlackBarDetector

Signature

```
role BlackBarDetector {
    UInt32                  firstLine;
    UInt32                  lastLine;
    UInt32                  vertRes;
    uhBbarDet_ValStat_t     status;
}
```

Qualifiers

— root

Description

The `BlackBarDetector` role determines the position of black bars in a video signal.

Independent Attributes

Table 1050

Name	Description
`firstLine`	The current first (non-black) video line as determined by the algorithm.
`lastLine`	The current last (non-black) video line as determined by the algorithm.
`vertRes`	The vertical resolution, that together with `firstLine` and `lastLine` defines the black bars. Note that this only equals the current vertical resolution of the input stream, if the algorithm that is used, immediately concludes that there are new first and last lines when the vertical input resolution changes.
`status`	Validity state of the black bar properties (first line, last line, vertical resolution).

Dependent Attributes

None.

Invariants

— 0 **<=** `firstLine` **<=** `lastLine` **<** `vertRes`.

Instantiation

See subclause 8.2.5.3.3.

Streaming Behavior

The `BlackBarDetector` has a video signal as an input. It determines the first and last (non-black) video lines of the input video. The `BlackBarDetector` does not change the input stream.

Active Behavior

The `BlackBarDetector` has standard notification behavior (see `uhIBbarDetNtf`).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.5.4.2 BlackBarDetectionClient

Signature
```
role BlackBarDetectionClient {}
```

Qualifiers
— actor

Description
A `BlackBarDetectionClient` represents a user of the `uhIBbarDet` interface.

8.2.5.5 Interfaces

8.2.5.5.1 uhIBbarDet

Qualifiers
None.

Description
This interface provides means to get the position of black bars in the input video and to reset the black bar detection algorithm.

Interface ID
uuid(486FAF36-49ED-4dfd-AD61-DE747770E195)

Remarks
In the following function specifications, the name `bbd` will be used to refer to the provider of this interface; see subclause 8.2.5.3.3.1.

8.2.5.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIBbarDetNtf               *pINotify,
    [in] UInt32                       cookie,
    [in] uhBbarDet_Notifications_t    notifs );
```

Qualifiers
— subscribe-function

8.2.5.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIBbarDetNtf               *pINotify,
    [in] UInt32                       cookie,
    [in] uhBbarDet_Notifications_t    notifs );
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers
— unsubscribe-function

8.2.5.5.1.3 GetFirstAndLastVideoLines

Signature
```
uhErrorCode_t GetFirstAndLastVideoLines (
    [out] puhBbarDet_ValStat_t pStatus,
    [out] pUInt32              pFirstLine,
    [out] pUInt32              pLastLine,
    [out] pUInt32              pVertRes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the first and last (non-black) video lines as determined by the algorithm. It also returns the vertical resolution, which is needed to determine the size of the black bars.

Parameters

Table 1051

Name	Description
`pStatus`	Output parameter that indicates the validity of the other [out] parameters.
`pFirstLine`	Output parameter returning the first (non-black) video line (determined by the algorithm).
`pLastLine`	Output parameter returning the last (non-black) video line (determined by the algorithm).
`pVertRes`	Output parameter returning the vertical resolution (frame based).

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pStatus == bbd.status`
— `*pFirstLine == bbd.firstLine`
— `*pLastLine == bbd.lastLine`
— `*pVertRes == bbd.vertRes`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.5.5.1.4 Reset

Signature
`uhErrorCode_t Reset ( );`

Qualifiers
— synchronous

— single-threaded

Description
Resets the black bar detection algorithm.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Action
— Modify `bbd.vertRes`
— Modify `bbd.firstLine`
— Modify `bbd.lastLine`

Postcondition
— `bbd.vertRes` == vertical resolution of the input
— `bbd.firstLine` == 0
— `bbd.lastLine` == `bbd.vertRes-1`

8.2.5.5.2 uhIBbarDetNtf

Qualifiers
— callback

Description
This is the notification interface to be provided by clients of the BBD that wish to subscribe to notifications of events of the BBD.

Interface ID
`uuid( 65199AF1-1787-4c84-8D86-5C3EEB1A1071 )`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.5.5.2.1 OnSubscriptionChanged

Signature

```
[async] Void OnSubscriptionChanged (
    [in] UInt32                         cookie,
    [in] uhBbarDet_Notifications_t      notifs );
```

Qualifiers

— onsubscriptionchanged-function

8.2.5.5.2.2 OnFirstAndLastVideoLinesChanged

Signature

```
[async] Void OnFirstAndLastVideoLinesChanged (
    [in] UInt32                                 cookie,
    [in] uhBbarDet_ValStat_t                    status,
    [in] UInt32                                 firstLine,
    [in] UInt32                                 lastLine,
    [in] UInt32                                 vertRes );
```

Qualifiers

None.

Description

This function is called by the `BlackBarDetector` whenever the algorithm signals a change of either the first or the last (non-black) video line (or both). Note that a particular algorithm may report new first and last lines some time after these have actually become effective. It is up to the algorithm to decide when it has done a stable detection.

Parameters

Table 1052

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	Parameter that indicates the validity of the other [in] parameters (first line, last line, vertical resolution).
`firstLine`	The first (non-black) video line.
`lastLine`	The last (non-black) video line.
`vertRes`	The vertical resolution (frame based).

Return Values

None.

Precondition

`True`

Action

— Any callback-compliant action.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`True`

8.2.6 UhlColEnh

8.2.6.1 Concepts

8.2.6.1.1 Introduction

Video Color Enhancement deals with enabling and disabling the color features Skin Tone Correction, Blue Stretch, and Green Enhancement.

Skin Tone Correction comes from the NTSC world, where the "hue" adjustment of the color decoder is critical. They eye is very sensitive to aberrations of skin tones. Colors that are possible skin tones are pulled towards a standard skin tone.



Before SkinTone Correction



After SkinTone Correction

Blue stretch comes from the Asian Pacific world, where very high color temperatures are typically appreciated. If we would just increase the gain of blue then everything would become bluish, including skin tones. Instead, only tints that are almost white will be made a very bright white. This will suggest a brighter picture, like it is the middle of a summer's day.



Before Blue Stretch



After Blue Stretch

Green enhancement increases the saturation of green tints. Grass and leaves will look like it is always springtime.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



8.2.6.2 Types & Constants

No additional types and constants are required for this logical component. For the basic types used, see Annex B of ISO/IEC 23004-1.

8.2.6.3 Logical Component

8.2.6.3.1 Interface-Role Model

Figure 118 depicts the interface-role model of the VCE (the grey box in Figure 118). It shows the control interface and the roles involved with the VCE.

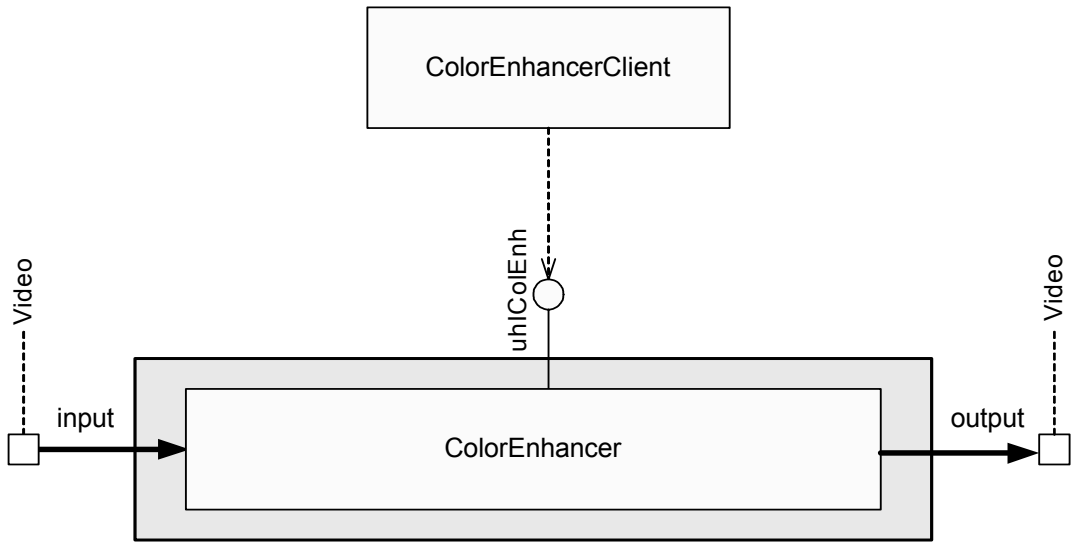


Figure 118 — Interface-Role Model

A VCE contains the role `ColorEnhancer` and provides the control interface `uhIColEnh`. A client (represented by the `ColorEnhancerClient` role) can use this control interface.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.6.3.2 Diversity

8.2.6.3.2.1 Provided Interfaces

Table 1053

Role	Interface	Presence
`ColorEnhancer`	`uhIColEnh`	mandatory

8.2.6.3.2.2 Configurable Items

None.

8.2.6.3.2.3 Constraints

— None

8.2.6.3.3 Instantiation

8.2.6.3.3.1 Objects Created

The following object is created when the logical component is instantiated:

Table 1054

Type	Object	Multiplicity
`ColorEnhancer`	`cenh`	1

8.2.6.3.3.2 Initial State

— `cenh.skinToneCorrectionEnabled == False`
— `cenh.blueStretchEnabled == False`
— `cenh.greenEnhancementEnabled == False`

8.2.6.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.6.4 Roles

8.2.6.4.1 ColorEnhancer

Signature

```
role ColorEnhancer {
    Bool skinToneCorrectionEnabled;
    Bool blueStretchEnabled;
    Bool greenEnhancementEnabled;
}
```

Qualifiers
Root.

Description
A `ColorEnhancer` enhances the colors of the incoming video signal. Supported operations are enabling or disabling the color features Skin Tone Correction, Blue Stretch, and Green Enhancement.

Independent Attributes

Table 1055

Name	Description
`skinToneCorrectionEnabled`	Indicates if the color feature Skin Tone Correction is enabled.
`blueStretchEnabled`	Indicates if the color feature Blue Stretch is enabled.
`greenEnhancementEnabled`	Indicates if the color feature Green Enhancement is enabled.

Dependent Attributes
None.

Invariants

None.

Instantiation
The `ColorEnhancer` role is always instantiated as part of a VCE.

Streaming Behavior
The `ColorEnhancer` observes the incoming video signal, enhances this based on the control input of the client, and streams the enhanced video signal to the output of the VCE.

Active Behavior
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.6.4.2 ColorEnhancerClient

Signature
`role ColorEnhancerClient {}`

Qualifiers
— actor

Description
A `ColorEnhancerClient` represents a user of the `uhIColEnh` interface. No constraints apply to the behavior of the `ColorEnhancerClient` other than those implied by Figure 1 — Simplified example graph..

8.2.6.5 Interfaces

8.2.6.5.1 uhIColEnh

Qualifiers
None.

Description
This interface of the `ColorEnhancer` role allows a `ColorEnhancerClient` to enable or disable the color features.

Interface ID
`uuid( 2800F840-AE38-4d6c-877A-B8BF5867F68B )`

8.2.6.5.1.1 GetSkinToneCorrectionEnabled

Signature
```
uhErrorCode_t GetSkinToneCorrectionEnabled ( [out] pBool pEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
Indicates if Skin Tone Correction is enabled (`True`) or disabled (`False`).

Parameters

Table 1056

Name	Description
`pEnabled`	Output parameter returning if Skin Tone Correction is enabled (`True`) or disabled (`False`).

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— None.

Postcondition
— `*`pEnabled `== cenh.skinToneCorrectionEnabled`

8.2.6.5.1.2 EnableSkinToneCorrection

Signature
```
uhErrorCode_t EnableSkinToneCorrection ( [in] Bool enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables (`True`) or disables (`False`) Skin Tone Correction.

Parameters

Table 1057

Name	Description
`enable`	Input parameter indicating whether to enable (`True`) or disable (`False`) Skin Tone Correction.

Return Values
Standard.

Precondition
`True`

Action
— Modify `cenh.skinToneCorrectionEnabled`.

Postcondition
— `cenh.skinToneCorrectionEnabled ==` *enable*

8.2.6.5.1.3 GetBlueStretchEnabled

Signature
```
uhErrorCode_t GetBlueStretchEnabled ( [out] pBool pEnabled );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Indicates if Blue Stretch is enabled (`True`) or disabled (`False`).

Parameters

Table 1058

Name	Description
`pEnabled`	Output parameter returning if Blue Stretch is enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*pEnabled == cenh.blueStretchEnabled`

8.2.6.5.1.4 EnableBlueStretch

Signature
```
uhErrorCode_t EnableBlueStretch ( [in] Bool enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables (`True`) or disables (`False`) Blue Stretch.

Parameters

Table 1059

Name	Description
`enable`	Input parameter indicating whether to enable (`True`) or disable (`False`) Blue Stretch.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `cenh.blueStretchEnabled`.

Postcondition
— `cenh.blueStretchEnabled == enable`

8.2.6.5.1.5 GetGreenEnhancementEnabled

Signature
```
uhErrorCode_t GetGreenEnhancementEnabled ( [out] pBool pEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
Indicates if Green Enhancement is enabled (`True`) or disabled (`False`).

Parameters

Table 1060

Name	Description
`pEnabled`	Output parameter returning if Green Enhancement is enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*`pEnabled `== cenh.greenEnhancementEnabled`

8.2.6.5.1.6 EnableGreenEnhancement

Signature
```
uhErrorCode_t EnableGreenEnhancement ( [in] Bool enable );
```

Qualifiers
— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Enables (`True`) or disables (`False`) Green Enhancement.

Parameters

Table 1061

Name	Description
`enable`	Input parameter indicating whether to enable (`True`) or disable (`False`) Green Enhancement.

Return Values
Standard.

Precondition
`True`

Action
— Modify `cenh.greenEnhancementEnabled`.

Postcondition
— `cenh.greenEnhancementEnabled == enable`

8.2.7 UhlCti

8.2.7.1 Concepts

8.2.7.1.1 Introduction

The UV bandwidth of an analog video signal sampled at 4:2:2 is considerably smaller than that of a video signal from a digital source. After up-sampling to 4:4:4, the UV spectrum of the former is 90% empty. To increase the perceived color sharpness and reduce the quality difference between analog and digital signals, one can perform color transient improvement (CTI), which steepens the transients present in the UV signal, thereby increasing the bandwidth. See Figure 36 for the effect of CTI.

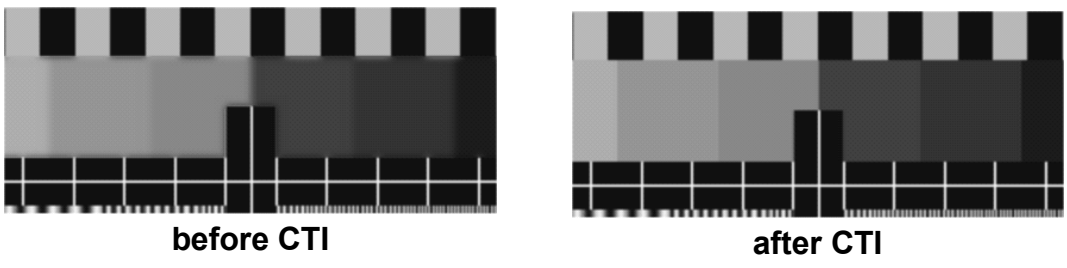


Figure 119 — The effect of CTI on a video signal.

The client can enable, disable, and set different levels of CTI. A higher level of CTI results in sharper color transitions. Note that for a real digital signal (from a digital source), CTI should be disabled or a lower level of CTI should be selected.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.7.2 Types & Constants

8.2.7.2.1 Public Types & Constants

8.2.7.2.1.1 uhCti_Notification_t

Signature

```
typedef enum _uhCti_Notification_t {
    uhCti_SubscriptionChanged = 0x00000001,
    uhCti_LevelChanged        = 0x00000002
} uhCti_Notification_t, *puhCti_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhICtiNtf`.

Values

Table 1062

Name	Description
`uhCti_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhCti_LevelChanged`	Represents `OnCtiLevelChanged`.

8.2.7.2.1.2 uhCti_Notifications_t

Signature

```
typedef Int32 uhCti_Notifications_t, *puhCti_Notifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type

uhCti_Notifications_t. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.2.7.2.1.3 uhCti_AllNotifications

Signature

```
const uhCti_Notifications_t uhCti_AllNotifications = 0x00000003;
```

Qualifiers

— None.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set of all values of type `uhCti_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.2.7.2.2 Model Types & Constants

None.

8.2.7.3 Logical Component

8.2.7.3.1 Interface-Role Model

The interface role-model is depicted in Figure 37

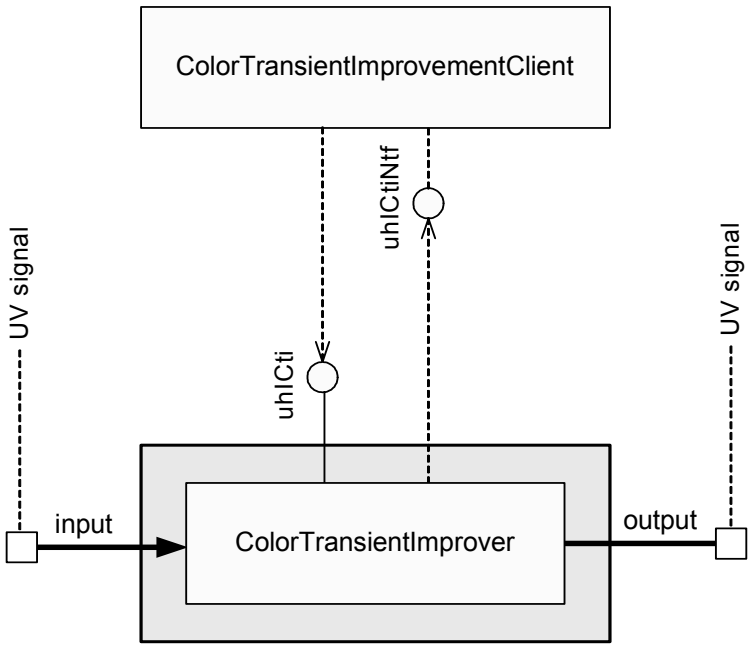


Figure 120 — Interface-Role Model of the Color Transient Improvement logical component

The Color Transient Improvement logical component (grey box) consists of the `ColorTransientImprover` role. This role provides an interface (`uhICti`) for controlling CTI. The control interface is meant for use by the client (`AudioAutomaticVolumeLevelingClient`). This client can optionally provide a call-back interface (`uhICtiNtf`) allowing it to subscribe to notifications of events in the Color Transient Improvement logical component.

8.2.7.3.2 Diversity

8.2.7.3.2.1 Provided Interfaces

Table 1063

Role	Interface	Presence
`ColorTransientImprover`	`uhICti`	mandatory
`ColorTransientImprovementClient`	`uhICtiNtf`	optional

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.7.3.2.2 Configurable Items

Table 1064

Role	Attribute
ColorTransientImprover	minLevel
ColorTransientImprover	maxLevel

8.2.7.3.2.3 Constraints

```
maxLevel >= minLevel
```

8.2.7.3.3 Instantiation

8.2.7.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1065

Type	Object	Multiplicity
ColorTransientImprover	cti	1

8.2.7.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
— cti.enabled == False
— cti.level   == cti.minLevel
```

8.2.7.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.7.4 Roles

8.2.7.4.1 ColorTransientImprover

Signature

```
role ColorTransientImprover {
    Bool                enabled;
    const UInt32        minLevel;
    const UInt32        maxLevel;
    UInt32              level;
}
```

Qualifiers

— root

Description

The `ColorTransientImprover` role controls the CTI level for enhancing the steepness of transients in the UV component of a video stream. The level of CTI can be set and the Color Transient Improver can be enabled or disabled.

Independent Attributes

Table 1066

Name	Description
`enabled`	Indicates whether CTI is enabled or not.
`minLevel`	Minimum value for the CTI level.
`maxLevel`	Maximum value for the CTI level.
`level`	Current value of the CTI level.

Dependent Attributes

None.

Invariants

— `minLevel <=` level `<= maxLevel`

Instantiation

See subclause 7.2.1.3.3.

Streaming Behavior

The `ColorTransientImprover` has a UV signal as an input. When the `ColorTransientImprover` is enabled, it will perform CTI on the input stream. This processed stream is passed to the output. When the `ColorTransientImprover` is disabled, the stream at the input is passed to the output without any processing.

Active Behavior

The `ColorTransientImprover` has standard notification behavior (see `uhICtiNtf`).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.7.4.2 ColorTransientImprovementClient

Signature
```
role ColorTransientImprovementClient {}
```

Qualifiers
— actor

Description
A `AudioAutomaticVolumeLevelingClient` represents a user of the `uhICti` interface.

8.2.7.5 Interfaces

8.2.7.5.1 uhICti

Qualifiers
None.

Description
This interface controls the level of CTI that is applied to the video signal.

Interface ID
```
uuid( A91557A5-8DFF-41ce-873F-BD9401AB94C8 )
```

Remarks
In the following function specifications, the name `cti` will be used to refer to the provider of this interface; see subclause 8.2.7.3.3.1.

8.2.7.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhICtiNtf               *pINotify,
    [in] UInt32                  cookie,
    [in] uhCti_Notifications_t   notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.7.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhICtiNtf               *pINotify,
    [in] UInt32                  cookie,
    [in] uhCti_Notifications_t   notifs );
```

Qualifiers

— unsubscribe-function

8.2.7.5.1.3 GetEnabled

Signature

```
uhErrorCode_t GetEnabled (
    [out] pBool       pEnabled);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether CTI is enabled or not.

Parameters

Table 1067

Name	Description
`pEnabled`	Output parameter returning whether CTI is enabled (`True`) or disabled (`False`).

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— **pEnabled* `== cti.enabled`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.7.5.1.4 Enable

Signature

```
uhErrorCode_t Enable (
    [in] Bool        enable );
```

Qualifiers

— synchronous

— single-threaded

Description

Enables or disables CTI.

Parameters

Table 1068

Name	Description
`enable`	Boolean to enable (`True`) or disable (`False`) CTI.

Return Values

Standard.

Precondition

`True`

Action

— Modify `cti.enabled`

Postcondition

— `cti.enabled ==` enable

8.2.7.5.1.5 GetLevel

Signature

```
uhErrorCode_t GetLevel (
    [out] pUInt32             pLevel );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the current CTI level.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1069

Name	Description
`pLevel`	Output parameter returning the current CTI level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— *`*pLevel == cti.level`*

8.2.7.5.1.6 SetLevel

Signature
```
uhErrorCode_t SetLevel (
    [in] UInt32           level);
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the CTI level.

Parameters

Table 1070

Name	Description
`level`	The CTI level to be set.

Return Values
Standard.

Precondition
— `cti.minLevel <=` level `<= cti.maxLevel`

Action
— Modify `cti.level`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `cti.level == level`

8.2.7.5.1.7 GetLevelRange

Signature
```
uhErrorCode_t GetLevelRange (
    [out] pUInt32       pMinLevel,
    [out] pUInt32       pMaxLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum value for the CTI level.

Parameters

Table 1071

Name	Description
`pMinLevel`	Output parameter returning the minimum value for the CTI level.
`pMaxLevel`	Output parameter returning the maximum value for the CTI level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMinLevel == cti.minLevel`
— `*pMaxLevel == cti.maxLevel`

8.2.7.5.2 uhICtiNtf

Qualifiers
— callback

Description
This is the notification interface to be provided by clients of the Color Transient Improvement logical component that wish to subscribe to notifications of events of the Color Transient Improvement logical component.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 0A2B13BA-033E-4b7e-9934-9D51A387CEB0 )`

8.2.7.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                  cookie,
    [in] uhCti_Notifications_t   notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.7.5.2.2 OnCtiLevelChanged

Signature
```
[async] Void OnCtiLevelChanged (
    [in] UInt32      cookie,
    [in] UInt32      level);
```

Qualifiers
None.

Description
This function is called by the `ColorTransientImprover` whenever it detects a change of the CTI level.

Table 1072

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`level`	The new CTI level.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.8 UhlDnr

8.2.8.1 Concepts

8.2.8.1.1 Dynamic Noise Reduction

This document defines the logical component Dynamic Noise Reducer. The Dynamic Noise Reducer logical component gives control over Dynamic Noise Reduction (DNR) functionality of the used platform. The following paragraph gives a short explanation of DNR.

Video pictures are often quite static. There is a lot of correlation between successive pictures. Additive noise is usually random (“Gaussian” noise), moving and not static. This makes the noise quite obvious to the eye. By averaging a series of pictures, the original picture content remains the same because of the correlation between the pictures. The random noise content is uncorrelated and will be reduced according to the number of pictures averaged. Temporal noise reduction will clean up the picture without affecting its resolution as long as the picture is static.

The principle of DNR is based on:

— strong temporal correlation in the picture,

— weak or no temporal correlation in the noise.

DNR increases the temporal correlation in the picture by temporal low-pass filtering. On a moving picture this will cause smearing. The effect is very similar to that of a long-persistence phosphor in certain picture tubes. This is of course not desired. So, in the presence of motion the noise reduction must be switched off. Two kinds of temporal variation must be distinguished: noise and motion. The threshold for this distinction may be set according to the current noise level. This is then called “Adaptive” Dynamic Noise Reduction.

There is another way to reduce the amount of noise: averaging the signal value of a pixel with that of neighboring pixels of which one has the confidence that they have approximately the same value. Small differences are interpreted as noise and may be filtered. Large differences are assumed to be intended and must not be filtered.

Above, two different types of noise filtering are described: temporal and spatial.

Temporal noise filtering uses temporal neighbors: the values of the same pixel in the past (or future). Motion will destroy the confidence in the historic value. Continued temporal filtering in the presence of motion causes motion smear.

Spatial noise filtering uses spatial neighbors: the value of pixels in the immediate vicinity. Strong details will destroy the confidence in neighbor values. Continued spatial filtering in the presence of detail causes loss of sharpness, plastic faces etc. There is of course a link with detail enhancement sharpness.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.8.2 Types & Constants

8.2.8.2.1 Public Types & Constants

8.2.8.2.1.1 uhDnr_Notification_t

Signature

```
typedef enum _uhDnr_Notification_t {
    uhDnr_SubscriptionChanged = 0x00000001,
    uhDnr_LevelChanged        = 0x00000002
} uhDnr_Notification_t, *puhDnr_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIDnrNtf`.

Values

Table 1073

Name	Description
`uhDnr_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhDnr_LevelChanged`	Represents `OnDnrLevelChanged`.

8.2.8.2.1.2 uhDnr_Notifications_t

Signature

```
typedef Int32 uhDnr_Notifications_t, *puhDnr_Notifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`Dnr_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.2.8.2.1.3 uhDnr_AllNotifications

Signature

```
const uhDnr_Notifications_t uhDnr_AllNotifications = 0x00000003;
```

Qualifiers

— None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set of all values of type uh`Dnr_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.2.8.2.2 Model Types & Constants

None.

8.2.8.3 Logical Component

8.2.8.3.1 Interface-Role Model

The interface role-model is depicted in Figure 121

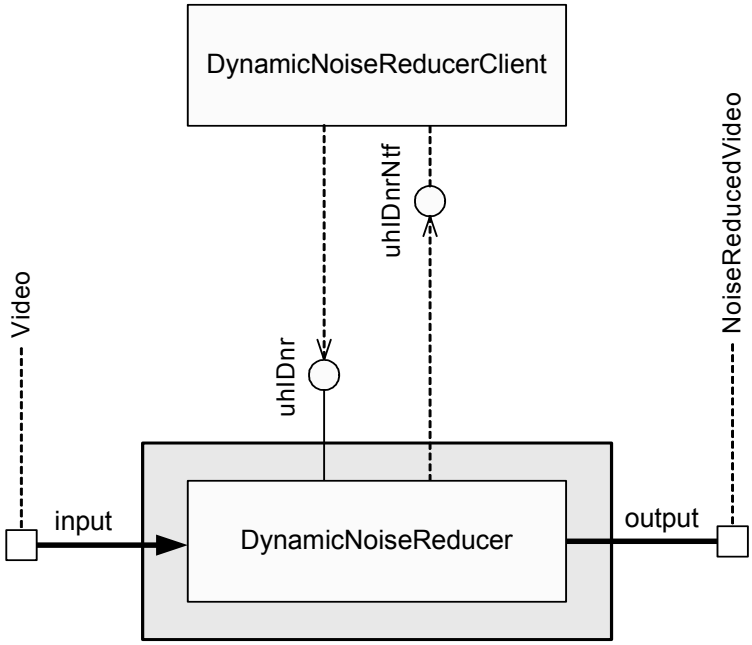


Figure 121 — Interface-Role Model of the DynamicNoiseReducer

The Dynamic Noise Reducer logical component (grey box) consists of the `DynamicNoiseReducer` role. This role provides an interface (`uhIDnr`) for controlling the dynamic noise reduction. The control interface is meant for use by the client (`DynamicNoiseReducerClient`). This client can optionally provide a call-back interface (`uhIDnrNtf`) allowing it to subscribe to notifications of events in the Dynamic Noise Reducer.

8.2.8.3.2 Diversity

8.2.8.3.2.1 Provided Interfaces

Table 1074

Role	Interface	Presence
`DynamicNoiseReducer`	`uhIDnr`	mandatory
`DynamicNoiseReducerClient`	`uhIDnrNtf`	optional

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.8.3.2.2 Configurable Items

Table 1075

Role	Attribute
DynamicNoiseReducer	minDnrLevel
DynamicNoiseReducer	maxDnrLevel

8.2.8.3.2.3 Constraints

```
maxDnrLevel >= minDnrLevel
```

8.2.8.3.3 Instantiation

8.2.8.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1076

Type	Object	Multiplicity
DynamicNoiseReducer	dnr	1

8.2.8.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
— dnr.dnrEnabled == False
— dnr.dnrLevel   == dnr.minDnrLevel
```

8.2.8.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.8.4 Roles

8.2.8.4.1 DynamicNoiseReducer

Signature

```
role DynamicNoiseReducer {
    Bool dnrEnabled;
    const UInt32 minDnrLevel;
    const UInt32 maxDnrLevel;
    UInt32 dnrLevel;
}
```

Qualifiers

— root

Description

The `DynamicNoiseReducer` role controls the DNR level for reducing the amount of noise in a video stream. The level of DNR can be set and the Dynamic Noise Reducer can be enabled or disabled.

Independent Attributes

Table 1077

Name	Description
`dnrEnabled`	Indicates whether DNR is enabled or not.
`minDnrLevel`	Minimum value for the DNR level.
`maxDnrLevel`	Maximum value for the DNR level.
`dnrLevel`	Current value of the DNR level.

Dependent Attributes

None.

Invariants

— minDnrLevel <= *dnrLevel* <= maxDnrLevel

Instantiation

See subclause 8.2.8.3.3.

Streaming Behavior

The `DynamicNoiseReducer` has a video stream as an input. When the `DynamicNoiseReducer` is enabled, it will try to reduce the amount of noise in the stream. This processed stream is passed to the output. When the `DynamicNoiseReducer` is disabled, the stream at the input is passed to the output without any processing.

Active Behavior

The `DynamicNoiseReducer` has standard notification behavior (see `uhIDnrNtf`).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.8.4.2 DynamicNoiseReducerClient

Signature
```
role DynamicNoiseReducerClient {}
```

Qualifiers
— actor

Description
A `DynamicNoiseReducerClient` represents a user of the `uhIDnr` interface.

8.2.8.5 Interfaces

8.2.8.5.1 uhIDnr

Qualifiers
None.

Description
This interface controls the level of DNR (Dynamic Noise Reduction) that is applied to the video signal.

Interface ID
```
uuid( 5D59534C-990E-470b-9487-A8C44F3B9B7B )
```

Remarks
— In the following function specifications, the name dnr will be used to refer to the provider of this interface; see subclause 8.2.8.3.3.1.

8.2.8.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIDnrNtf               *pINotify,
    [in] UInt32                  cookie,
    [in] uhDnr_Notifications_t   notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.8.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIDnrNtf               *pINotify,
    [in] UInt32                  cookie,
    [in] uhDnr_Notifications_t   notifs );
```

Qualifiers

— unsubscribe-function

8.2.8.5.1.3 GetEnabled

Signature

```
uhErrorCode_t GetEnabled (
    [out] pBool pDnrEnabled);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether DNR is enabled or not.

Parameters

Table 1078

Name	Description
`pDnrEnabled`	Output parameter returning whether DNR is enabled (`True`) or disabled (`False`).

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— *`*pDnrEnabled`* `== dnr.dnrEnabled`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.8.5.1.4 Enable

Signature
```
uhErrorCode_t Enable (
    [in] Bool dnrEnabled );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables DNR.

Parameters

Table 1079

Name	Description
`dnrEnabled`	Boolean to enable (`True`) or disable (`False`) the dynamic noise reduction.

Return Values
Standard.

Precondition
`True`

Action
— Modify `dnr.dnrEnabled`

Postcondition
— `dnr.dnrEnabled ==` dnrEnabled

8.2.8.5.1.5 GetLevel

Signature
```
uhErrorCode_t GetLevel (
    [out] pUInt32 pDnrLevel );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current DNR level.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1080

Name	Description
`pDnrLevel`	Output parameter returning the current DNR level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— *`*pDnrLevel` == `dnr.dnrLevel`*

8.2.8.5.1.6 SetLevel

Signature
```
uhErrorCode_t SetLevel (
    [in] UInt32 dnrLevel);
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the DNR level.

Parameters

Table 1081

Name	Description
`dnrLevel`	The DNR level to be set.

Return Values
Standard.

Precondition
— `dnr.minDnrLevel <=` dnrLevel `<= dnr.maxDnrLevel`

Action
— Modify `dnr.dnrLevel`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`dnr.dnrLevel == dnrLevel`

8.2.8.5.1.7 GetLevelRange

Signature
```
uhErrorCode_t GetLevelRange (
    [out] pUInt32 pMinDnrLevel,
    [out] pUInt32 pMaxDnrLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum value for the DNR level.

Parameters

Table 1082

Name	Description
`pMinDnrLevel`	Output parameter returning the minimum value for the DNR level.
`pMaxDnrLevel`	Output parameter returning the maximum value for the DNR level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*`*pMinDnrLevel* `== dnr.minDnrLevel`
— `*`pMaxDnrLevel `== dnr.maxDnrLevel`

8.2.8.5.2 uhIDnrNtf

Qualifiers
— callback

Description
This is the notification interface to be provided by dynamic noise reducer clients that wish to subscribe to notifications of dynamic noise reducer events.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 6ED768AB-54D8-44bc-993A-638F60FCD555 )`

8.2.8.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                  cookie,
    [in] uhDnr_Notifications_t   notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.8.5.2.2 OnDnrLevelChanged

Signature
```
[async] Void OnDnrLevelChanged (
    [in] UInt32 cookie,
    [in] UInt32 dnrLevel);
```

Qualifiers
None.

Description
This function is called by the `DynamicNoiseReducer` whenever it detects a change of the DNR level.

Table 1083

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`dnrLevel`	The new DNR level.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.9 UhlHistoMod

8.2.9.1 Concepts

8.2.9.1.1 Introduction

Histogram Modification (HM) can be used to improve the local contrast in a picture of which the distribution of grey levels is sub-optimal. First, the histogram of the picture is measured: the luminance range is divided in equal intervals, and for each interval the pixels with a luminance value falling in that interval are counted. From such a histogram, a transfer curve can be calculated that adjusts the luminance[24] of the pixels in such a way that the histogram of the adjusted picture is somewhat flatter than the original histogram (see Figure 122 for an example with 5 intervals), which is perceived as a better quality picture.

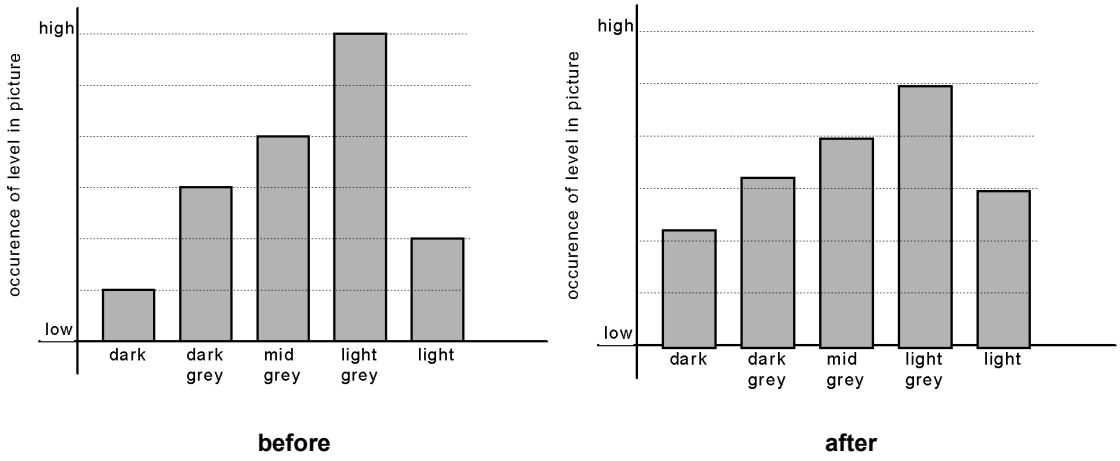


Figure 122 — Example of the luminance histogram of a TV picture (before and after histogram modification).

The client can enable, disable, and set different levels of HM. A higher level of HM results in more contrast.

8.2.9.2 Types & Constants

8.2.9.2.1 Public Types & Constants

8.2.9.2.1.1 uhHistoMod_Notification_t

Signature

```
typedef enum _uhHistoMod_Notification_t {
    uhHistoMod_SubscriptionChanged = 0x00000001,
    uhHistoMod_LevelChanged        = 0x00000002
} uhHistoMod_Notification_t, *puhHistoMod_Notification_t;
```

Qualifiers

— enum-element

[24] The chrominance (UV) component of the video signal is also adjusted by HM.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIHistoModNtf`.

Values

Table 1084

Name	Description
`uhHistoMod_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhHistoMod_LevelChanged`	Represents `OnHistoModLevelChanged`.

8.2.9.2.1.2 uhHistoMod_Notifications_t

Signature
```
typedef Int32 uhHistoMod_Notifications_t, *puhHistoMod_Notifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`HistoMod_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

8.2.9.2.1.3 uhHistoMod_AllNotifications

Signature
```
const uhHistoMod_Notifications_t uhHistoMod_AllNotifications = 0x00000003;
```

Qualifiers
— None.

Description
Defines the set of all values of type uh`HistoMod_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.2.9.2.2 Model Types & Constants

None.

8.2.9.3 Logical Component

8.2.9.3.1 Interface-Role Model

The interface role-model is depicted in Figure 123

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

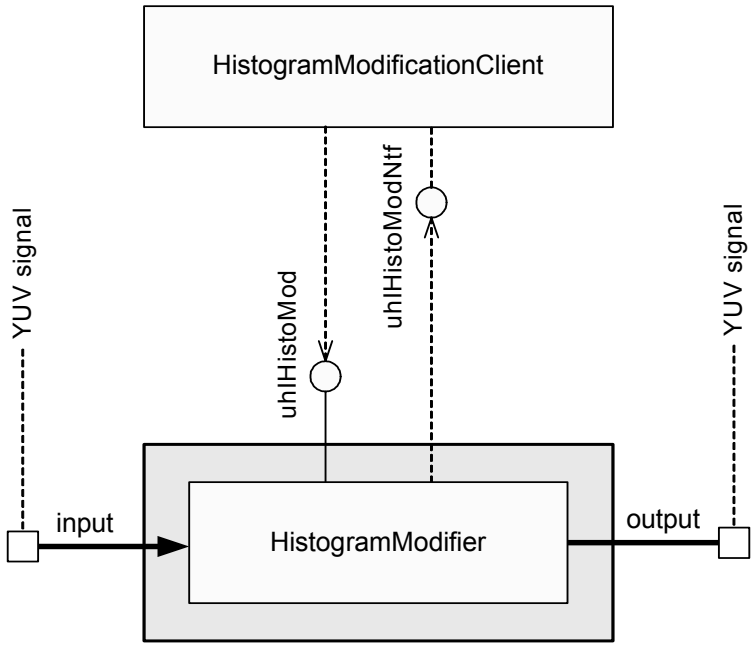


Figure 123 — Interface-Role Model of the Histogram Modification logical component

The Histogram Modification logical component (grey box) consists of the `HistogramModifier` role. This role provides an interface (`uhIHistoMod`) for controlling HM. The control interface is meant for use by the client (`HistogramModificationClient`). This client can optionally provide a call-back interface (`uhIHistoModNtf`) allowing it to subscribe to notifications of events in the Histogram Modification logical component.

8.2.9.3.2 Diversity

8.2.9.3.2.1 Provided Interfaces

Table 1085

Role	Interface	Presence
`HistogramModifier`	`uhIHistoMod`	mandatory
`HistogramModificationClient`	`uhIHistoModNtf`	optional

8.2.9.3.2.2 Configurable Items

Table 1086

Role	Attribute
`HistogramModifier`	`minLevel`
`HistogramModifier`	`maxLevel`

8.2.9.3.2.3 Constraints

— maxLevel >= *minLevel*

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.9.3.3 Instantiation

8.2.9.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1087

Type	Object	Multiplicity
`HistogramModifier`	`hm`	1

8.2.9.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
hm.enabled == False

hm.level   == hm.minLevel
```

8.2.9.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.2.9.4 Roles

8.2.9.4.1 HistogramModifier

Signature

```
role HistogramModifier {
    Bool             enabled;
    const UInt32     minLevel;
    const UInt32     maxLevel;
    UInt32           level;
}
```

Qualifiers

— root

Description

The `HistogramModifier` role controls the HM level for enhancing the contrast of a video signal. The level of HM can be set and the `HistogramModifier` can be enabled or disabled.

Independent Attributes

Table 1088

Name	Description
`enabled`	Indicates whether HM is enabled or not.
`minLevel`	Minimum value for the HM level.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
`maxLevel`	Maximum value for the HM level.
`level`	Current value of the HM level.

Dependent Attributes
None.

Invariants

```
minLevel <= level <= maxLevel
```

Instantiation
See subclause 8.2.9.3.3.

Streaming Behavior
The `HistogramModifier` has a YUV signal as an input. When the `HistogramModifier` is enabled, it will perform HM on the input stream. This processed stream is passed to the output. When the `HistogramModifier` is disabled, the stream at the input is passed to the output without any processing.

Active Behavior
The `HistogramModifier` has standard notification behavior (see `uhIHistoModNtf`).

8.2.9.4.2 HistogramModificationClient

Signature
`role` `HistogramModificationClient {}`

Qualifiers
— actor

Description
A `HistogramModificationClient` represents a user of the `uhIHistoMod` interface.

8.2.9.5 Interfaces

8.2.9.5.1 uhIHistoMod

Qualifiers
None.

Description
This interface controls the level of HM that is applied to the video signal.

Interface ID
`uuid( 2E687DB9-E887-41b8-B014-165F95D38613 )`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks

— In the following function specifications, the name hm will be used to refer to the provider of this interface; see subclause 8.2.9.3.3.1.

8.2.9.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIHistoModNtf               *pINotify,
    [in] UInt32                       cookie,
    [in] uhHistoMod_Notifications_t   notifs );
```

Qualifiers

— subscribe-function

8.2.9.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIHistoModNtf               *pINotify,
    [in] UInt32                       cookie,
    [in] uhHistoMod_Notifications_t   notifs );
```

Qualifiers

— unsubscribe-function

8.2.9.5.1.3 GetEnabled

Signature

```
uhErrorCode_t GetEnabled (
    [out] pBool       pEnabled);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether HM is enabled or not.

Parameters

Table 1089

Name	Description
`pEnabled`	Output parameter returning whether HM is enabled (`True`) or disabled (`False`).

Return Values

Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
— `*`pEnabled `== hm.enabled`

8.2.9.5.1.4 Enable

Signature
```
uhErrorCode_t Enable (
    [in] Bool        enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables HM.

Parameters

Table 1090

Name	Description
`enable`	Boolean to enable (`True`) or disable (`False`) HM.

Return Values
Standard.

Precondition
`True`

Action
— Modify `hm.enabled`

Postcondition
— `hm.enabled` **==** ***enable***

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.9.5.1.5 GetLevel

Signature

```
uhErrorCode_t GetLevel (
    [out] pUInt32             pLevel );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the current HM level.

Parameters

Table 1091

Name	Description
`pLevel`	Output parameter returning the current HM level.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*pLevel == hm.level`

8.2.9.5.1.6 SetLevel

Signature

```
uhErrorCode_t SetLevel (
    [in] UInt32           level);
```

Qualifiers

— synchronous

— single-threaded

Description

Sets the HM level.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1092

Name	Description
`level`	The HM level to be set.

Return Values
Standard.

Precondition
— `hm.minLevel <= level <= hm.maxLevel`

Action
— Modify `hm.level`

Postcondition
— `hm.level ==` level

8.2.9.5.1.7 GetLevelRange

Signature

```
uhErrorCode_t GetLevelRange (
    [out] pUInt32       pMinLevel,
    [out] pUInt32       pMaxLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum value for the HM level.

Parameters

Table 1093

Name	Description
`pMinLevel`	Output parameter returning the minimum value for the HM level.
`pMaxLevel`	Output parameter returning the maximum value for the HM level.

Return Values
Standard.

Precondition
— True

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— **pMinLevel* == hm.minLevel
— *pMaxLevel == hm.maxLevel

8.2.9.5.2 uhlHistoModNtf

Qualifiers
— callback

Description
This is the notification interface to be provided by clients of the Histogram Modification logical component that wish to subscribe to notifications of events of the Histogram Modification logical component.

Interface ID
uuid(D89F166C-6DF0-4f9e-8364-79C7468679E6)

8.2.9.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                     cookie,
    [in] uhHistoMod_Notifications_t   notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.9.5.2.2 OnHistoModLevelChanged

Signature
```
[async] Void OnHistoModLevelChanged (
    [in] UInt32      cookie,
    [in] UInt32      level);
```

Qualifiers
None.

Description
This function is called by the HistogramModifier whenever it detects a change of the HM level.

Table 1094

Name	Description
cookie	The value that was passed by the client at subscription time.
level	The new HM level.

Return Values
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.2.10 UhlNoiseMeas

8.2.10.1 Concepts

8.2.10.1.1 Noise Measurement

The Noise Measurement logical component offers an abstract view on the amount of noise in a video signal. A discrete range of noise "levels" is defined by `minLevel` and `maxLevel`. The current noise level of the video signal is a number within that range, corresponding to the amount of noise in the signal (Figure 124). The meter scale defined by this range is a linear scale, so going from one level to the next always corresponds with (more or less) the same increase in noise.

The extremes of the meter scale, `minLevel` and `maxLevel`, do not necessarily correspond with an absolute minimum and an absolute maximum amount of noise, so, when the meter has reached `maxLevel` and the signal becomes noisier, the client won't notice it. The inverse holds for `minLevel`.

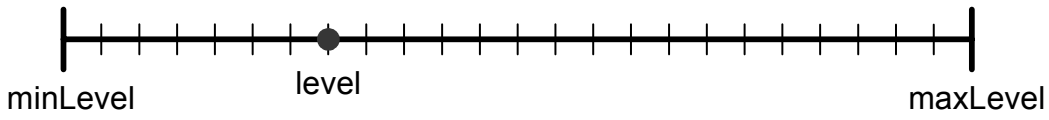


Figure 124 — A discrete linear meter scale.

8.2.10.2 Types & Constants

8.2.10.2.1 Public Types & Constants

8.2.10.2.1.1 uhNoiseMeas_ValStat_t

Signature
```
typedef enum _uhNoiseMeas_ValStat_t {
    uhNoiseMeas_ValStatInvalid = -2,
    uhNoiseMeas_ValStatUnknown = -1,
    uhNoiseMeas_ValStatKnown   =  1
} uhNoiseMeas_ValStat_t, *puhNoiseMeas_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1095

Name	Description
uhNoiseMeas_ValStatInvalid	The property/properties are temporarily not available.
uhNoiseMeas_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhNoiseMeas_ValStatKnown	The property/properties are available and determined.

8.2.10.2.1.2 uhNoiseMeas_Notification_t

Signature

```
typedef enum _uhNoiseMeas_Notification_t {
    uhNoiseMeas_SubscriptionChanged = 0x00000001,
    uhNoiseMeas_LevelChanged        = 0x00000002
} uhNoiseMeas_Notification_t, *puhNoiseMeas_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhINoiseMeasNtf`.

Values

Table 1096

Name	Description
uhNoiseMeas_SubscriptionChanged	Represents `OnSubscriptionChanged`.
uhNoiseMeas_LevelChanged	Represents `OnNoiseLevelChanged`.

8.2.10.2.1.3 uhNoiseMeas_Notifications_t

Signature

```
typedef Int32 uhNoiseMeas_Notifications_t, *puhNoiseMeas_Notifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`NoiseMeas_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.10.2.1.4 uhNoiseMeas_AllNotifications

Signature
```
const uhNoiseMeas_Notifications_t uhNoiseMeas_AllNotifications = 0x00000003;
```

Qualifiers
— None.

Description
Defines the set of all values of type uh`NoiseMeas_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.2.10.2.2 Model Types & Constants

None.

8.2.10.3 Logical Component

8.2.10.3.1 Interface-Role Model

The interface role-model is depicted in Figure 125

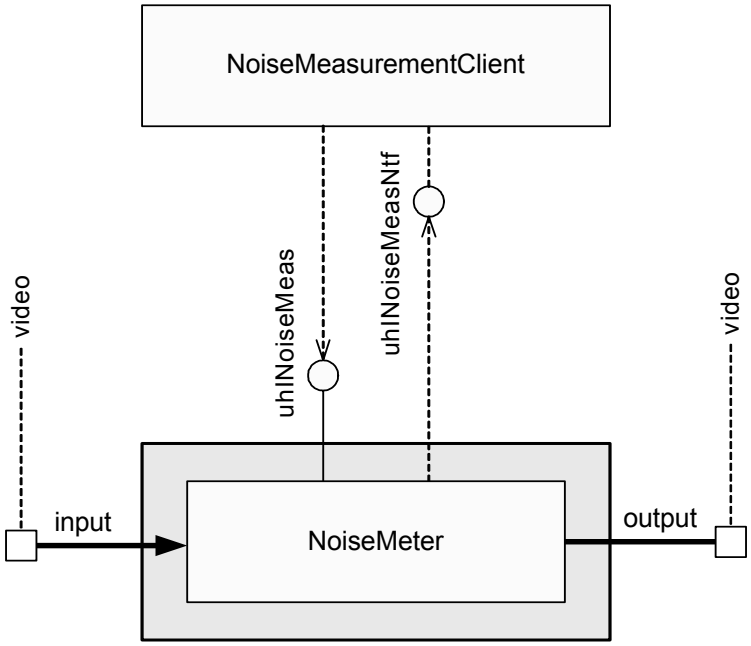


Figure 125 — Interface-Role Model of the Noise Measurement logical component

The Noise Measurement logical component (grey box) consists of the `NoiseMeter` role. This role provides an interface (`uhINoiseMeas`) for controlling the noise measurement. The control interface is meant for use by the client (`NoiseMeasurementClient`). This client can optionally provide a call-back interface (`uhINoiseMeasNtf`) allowing it to subscribe to notifications of events in the Noise Measurement logical component.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.10.3.2 Diversity

8.2.10.3.2.1 Provided Interfaces

Table 1097

Role	Interface	Presence
NoiseMeter	uhINoiseMeas	mandatory
NoiseMeasurementClient	uhINoiseMeasNtf	optional

8.2.10.3.2.2 Configurable Items

Table 1098

Role	Attribute
NoiseMeter	minLevel
NoiseMeter	maxLevel

8.2.10.3.2.3 Constraints

— maxLevel >= *minLevel*

8.2.10.3.3 Instantiation

8.2.10.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1099

Type	Object	Multiplicity
NoiseMeter	nm	1

8.2.10.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
— nm.status  == uhNoiseMeas_ValStatUnknown
— nm.enabled == False
— nm.level   == undefined
```

8.2.10.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.10.4 Roles

8.2.10.4.1 NoiseMeter

Signature

```
role NoiseMeter {
    uhNoiseMeas_ValStat_t status;
    Bool                  enabled;
    const UInt32          minLevel;
    const UInt32          maxLevel;
    UInt32                level;
}
```

Qualifiers

— root

Description

The `NoiseMeter` role provides means to get an indication of the amount of noise in a video signal. The noise level can be queried and the `NoiseMeter` can be enabled or disabled.

Independent Attributes

Table 1100

Name	Description
`status`	Validity state of the noise measurement property.
`enabled`	Indicates whether noise measurement is enabled or not.
`minLevel`	Minimum noise level.
`maxLevel`	Maximum noise level.
`level`	Current noise level.

Dependent Attributes

None.

Invariants

— minLevel <= *level* <= maxLevel

Instantiation

See subclause 8.2.10.3.3.

Streaming Behavior

The `NoiseMeter` has a video signal as an input. When the `NoiseMeter` is enabled, it will perform noise measurement on the input stream. When the `NoiseMeter` is disabled, the stream at the input is passed to the output without any measurement being done (level is not updated).

Active Behavior

The `NoiseMeter` has standard notification behavior (see `uhINoiseMeasNtf`).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.10.4.2 NoiseMeasurementClient

Signature
```
role NoiseMeasurementClient {}
```

Qualifiers
— actor

Description
A `NoiseMeasurementClient` represents a user of the `uhINoiseMeas` interface.

8.2.10.5 Interfaces

8.2.10.5.1 uhINoiseMeas

Qualifiers
None.

Description
This interface provides means to get an indication of the amount of noise in the video signal.

Interface ID
```
uuid( 3F28370B-1418-48fd-8BFE-D1BA35B6E25F )
```

Remarks
In the following function specifications, the name nm will be used to refer to the provider of this interface; see subclause 8.2.10.3.3.1.

8.2.10.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhINoiseMeasNtf              *pINotify,
    [in] UInt32                       cookie,
    [in] uhNoiseMeas_Notifications_t  notifs );
```

Qualifiers
— subscribe-function

8.2.10.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhINoiseMeasNtf              *pINotify,
    [in] UInt32                       cookie,
    [in] uhNoiseMeas_Notifications_t  notifs );
```

Qualifiers
— unsubscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.10.5.1.3 GetEnabled

Signature
```
uhErrorCode_t GetEnabled (
    [out] pBool       pEnabled);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether noise measurement is enabled or not.

Parameters

Table 1101

Name	Description
`pEnabled`	Output parameter returning whether noise measurement is enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*`*`pEnabled`* `== nm.enabled`

8.2.10.5.1.4 Enable

Signature
```
uhErrorCode_t Enable (
    [in] Bool        enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables noise measurement.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1102

Name	Description
`enable`	Boolean to enable (`True`) or disable (`False`) noise measurement.

Return Values
Standard.

Precondition
`True`

Action
— Modify `nm.enabled`

Postcondition
— `nm.enabled ==` enable

8.2.10.5.1.5 GetLevel

Signature

```
uhErrorCode_t GetLevel (
    [out] puhNoiseMeas_ValStat_t pStatus,
    [out] pUInt32                pLevel );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current noise level.

Parameters

Table 1103

Name	Description
`pStatus`	Output parameter that indicates the validity of the level parameter.
`pLevel`	Output parameter returning the current noise level.

Return Values
Standard.

Precondition
`True`

Action
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `*pStatus == nm.status`
— `*pLevel == nm.level`

8.2.10.5.1.6 GetLevelRange

Signature
```
uhErrorCode_t GetLevelRange (
    [out] pUInt32       pMinLevel,
    [out] pUInt32       pMaxLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum noise levels.

Parameters

Table 1104

Name	Description
`pMinLevel`	Output parameter returning the minimum noise level.
`pMaxLevel`	Output parameter returning the maximum noise level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMinLevel == nm.minLevel`
— `*pMinLevel == nm.maxLevel`

8.2.10.5.2 uhINoiseMeasNtf

Qualifiers
— callback

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This is the notification interface to be provided by clients of the Noise Measurement logical component that wish to subscribe to notifications of events of the Noise Measurement logical component.

Interface ID
`uuid( 1BD48979-C4F1-444d-8D20-63ADA3F20885 )`

8.2.10.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
   [in] UInt32                        cookie,
   [in] uhNoiseMeas_Notifications_t   notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.10.5.2.2 OnNoiseLevelChanged

Signature
```
[async] Void OnNoiseLevelChanged (
   [in] UInt32              cookie,
   [in] uhNoiseMeas_ValStat_t status,
   [in] UInt32              level);
```

Qualifiers
None.

Description
This function is called by the `NoiseMeter` whenever it detects a change of the noise level.

Table 1105

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	Parameter that indicates the validity of the level parameter.
`level`	The new noise level.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11 uhIScanRateConv

8.2.11.1 Concepts

8.2.11.1.1 Scan Rate Conversion

This document defines the Scan Rate Converter logical component (LC). The Scan Rate Converter LC gives control over scan rate conversion functionality. The following paragraphs give a short explanation of scan rate conversions and how to use them.

8.2.11.1.1.1 Background

Scan rate conversions are video conversions by which the scan rate of the video at the output has been changed compared to the scan rate at the input. The main reasons to do these conversions are to:

— Adapt the original scan rate, such that it matches the display capabilities.

— Improve the perceived picture quality.

In order to do scan rate conversion video streams need de-interlacing and/or temporal conversion.

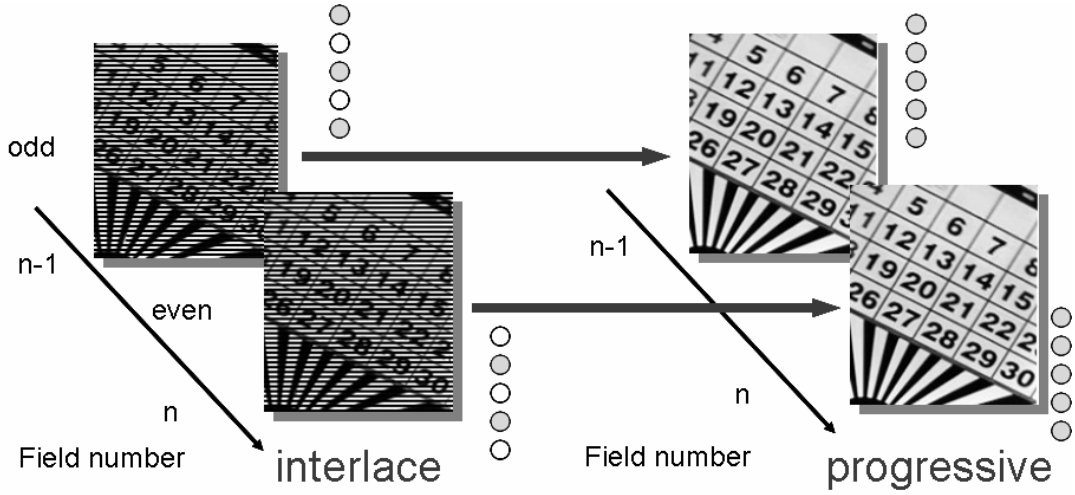


Figure 126 — Concept of de-interlacing

De-interlacing is the conversion of a interlaced picture to a progressive picture as depicted in Figure 126. De-interlacing algorithms vary from simple to very complex.

Selection of a de-interlacing algorithm mainly depends on:

— Content of source material:
De-interlacing algorithms often do well in a particular category (for example texture or robustness), but less in another.

— Source material:
Source material can be recorded by either a film or a video camera.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Film camera: A film camera shoots progressive pictures at 24 Hz. When broadcasted, the 24 Hz progressive material is interlaced (doubling the frequency) and speed-up to 50/60 Hz. On the receiving end the source material although having a frequency of 50/60 Hz, has motion information at 25/30 Hz as depicted in Figure 127.

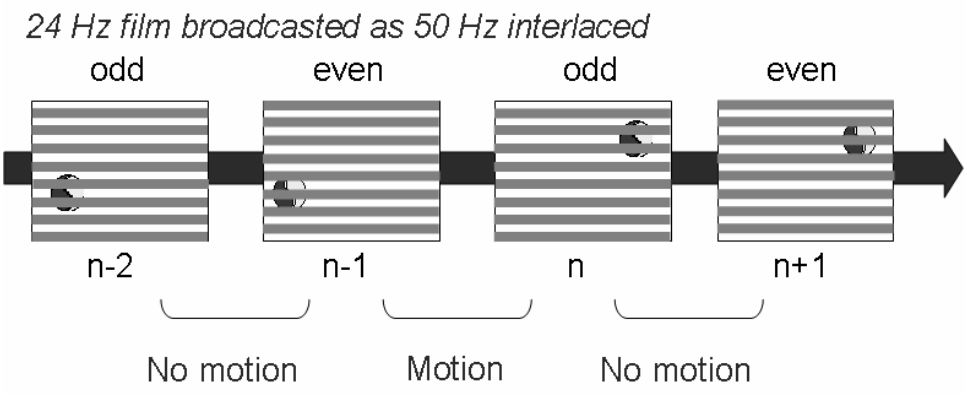


Figure 127 — Source material recorded with film camera.

Because there is no motion information between two interlaced fields out of the same progressive frame, de-interlacing source material recorded by a film camera is combining the correct interlaced fields to a progressive frame as depicted in Figure 128.

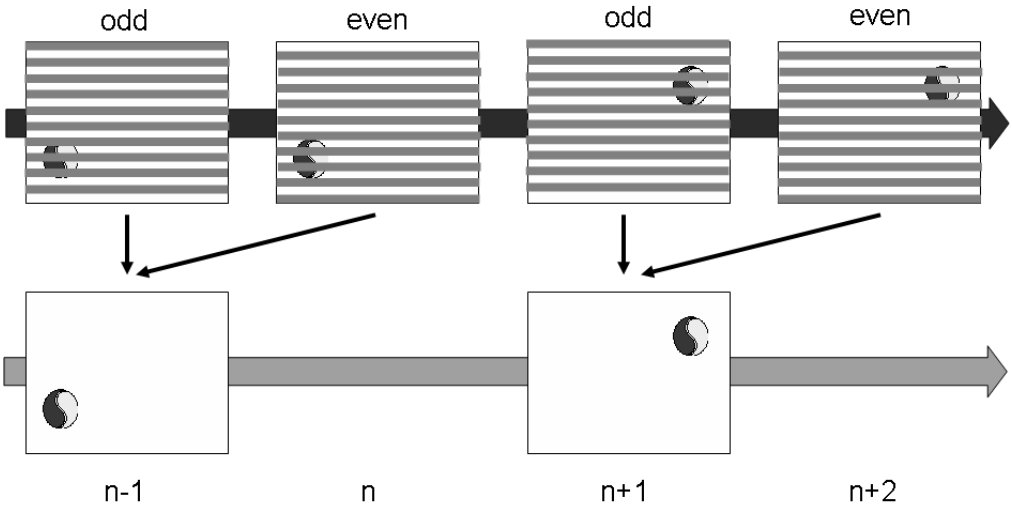


Figure 128 — De-interlacing film material.

— Video camera: A video camera shoots interlaced pictures at 50/60Hz, which directly can be broadcasted. Because there is motion information in between fields, combining to interlaced fields to one progressive frame can lead to unacceptable motion artifacts. For this reason de-interlacing source material recorded by a video camera often use more complex algorithms which need multi field input as depicted in Figure 129.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

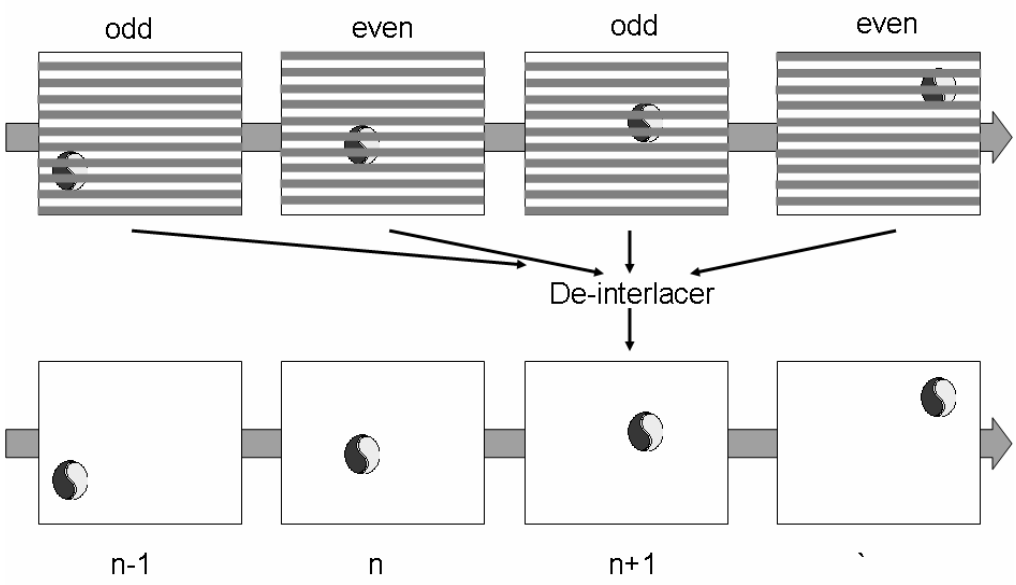


Figure 129 — De-interlacing video material.

Temporal conversion, can be either converting to a higher or lower temporal frequency by calculating the non-transmitted/recorded pictures in between the existing ones or replacing/skipping the temporal positions of the existing pictures.

Temporal conversions are done to:

— Improve video quality by reducing large area flicker.

— Drive a display that cannot handle the input field rate.

— Improve motion pattern in source material.

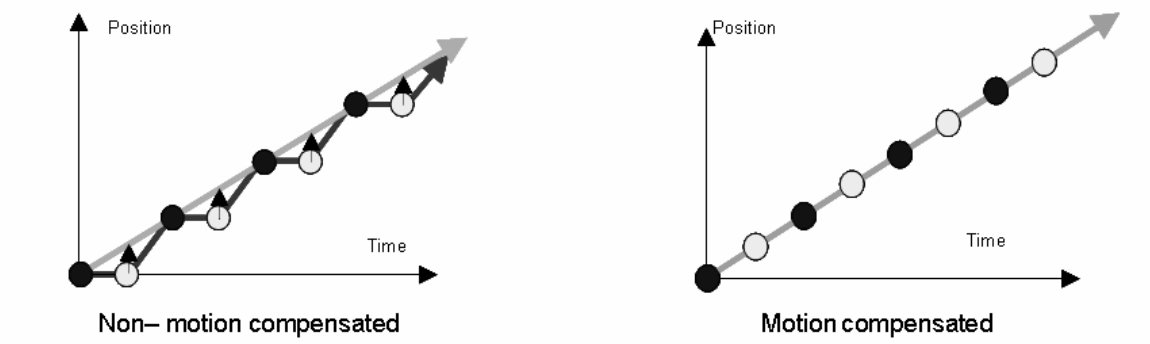


Figure 130 — Non-motion compensated versus motion compensated upconversion.

Temporal conversion can be done motion compensated or non-compensated as shown in Figure 130. Non-motion compensated temporal conversion (e.g. repetition of frames) leads to a distorted motion pattern. By compensating the intermediate frames the motion pattern can become natural again.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11.1.1.2 Usage

The scan rate converter interface offers control over the various algorithms to convert from an input scan rate to an output scan rate.

These algorithms control the following concepts:

a) De-interlacing

For de-interlacing several algorithms are available. Available algorithms can do processing in spatial or temporal domain, filtered or unfiltered, with or without motion compensation. Best choice of algorithm depends on source type and content of the source material but maybe restricted/limited by use-case (resource usage). De-interlacing algorithms often do well in a particular category (for example texture or robustness), but less in an other. For this reason the client can decide to leave the selection of an algorithm to the de-interlacer. In this case the de-interlacer will select, based on the source type and the content which algorithm is the best.

The need for de-interlacing depends on the relationship of the scan type between the input and the output video stream.

In case of progressive input de-interlacing is never needed.

- In case of interlaced input and progressive output, de-interlacing is always needed.
- In case of interlaced input and interlaced output, de-interlacing is optional. Internal processing may require progressive intermediates, by the platform may decide the input first has to be de-interlaced, processed and afterwards re-interlaced.

b) Scan/Motion rate conversion

Scan rate conversion is needed when there is a difference between the scan rate of the input format and the output format.

The motion rate represents the frequency of the motion information in the source material. This can differ from the actual field or frame rate. E.g. for film source in 50i or 50p, the motion rate is 25. If by de-interlacing or scan rate conversion the motion portrait is disturbed, this can be corrected via motion rate conversion.

c) Re-interlacing

In case the output scan type is interlaced and the input is progressive or a intermediate processing stage needs a progressive input, re-interlacing is needed. Re-interlacing simply consists of extracting even and odd fields from a progressive frame.

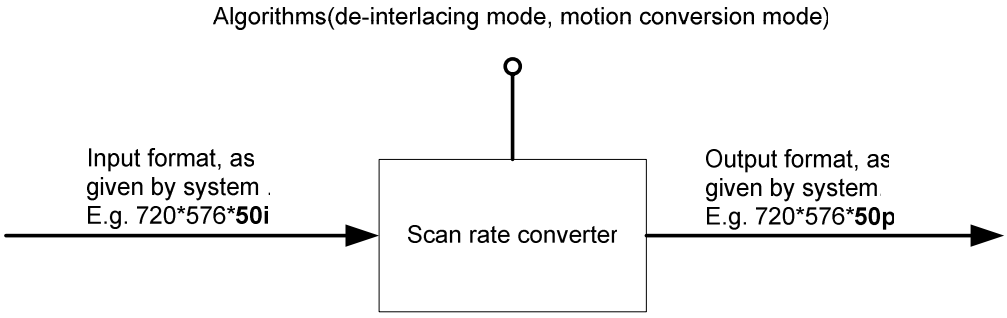


Figure 131 — Example concept scan rate conversion.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The scan rate converter depicted in Figure 131 can be controlled by:

— De-interlacing algorithms: Setting an algorithm used for re/de-interlacing. De-interlacing algorithms can vary from simple algorithms that only use information of the current field(spatial), to complex algorithms that use multiple fields(temporal), filters(e.g. average, median) and motion compensation. Table 1106 gives an example of the possible combinations.

Table 1106— Example combinations for de-interlacing

Algorithm	Temporal	Filtered	Motion compensated
Line repeat	False	False	False
Line filter	False	True	False
Field insert	True	**False**	False
Motion adaptive	True	True	False
Motion compensated	True	True	True
Auto motion adaptive	True	*Auto	False
Auto motion compensated	True	*Auto	True

* Auto: Depending on the source and motion type a selection is made. In case the source type is film or there is no motion in the source material, field insert is preferred and thus filters are disabled. In all other cases, filters will be enabled.

— Temporal conversion mode: Since the need for temporal conversion is determined by the ratio between the input and output format, the temporal conversion mode controls whether the conversion is done motion compensated or not.

Depending on ratio between input and output format and/or platform limitations, not all supported algorithms are always available. For this reason the scan rate converter maintains a list of available conversion algorithms. The client of the scan rate converter can subscribe on the changes of this list. In case the availability of one or more algorithms(s) changes the list is updated. If the updated list does not contain the algorithm as preferred by the client the scan rate converter uses a default algorithm, notifying the client of a change of the actual algorithm. The client then can choose to overrule by one of the list of available algorithms. The client can set a preferred algorithm at any time, as long as the preferred algorithm is a member of the available algorithms list it will reflect the actual algorithm.

8.2.11.1.1.3 Multiple Instantiation

The scan rate converter can be instantiated more than once. In case of multiple instantiation, system limitations cause a strong dependency between the different instantiations. In order to make mixing possible is it necessary that the inputs of the mixer have the same frequency and scan type. Because the scan rate converter instances are logically placed before the mixer, the output of the scan rate converter instances

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

should all be the same. For this reason the output format of the scan rate converter always represents the actual output format.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11.2 Types & Constants

8.2.11.2.1 Public Types & Constants

8.2.11.2.1.1 Error Codes

None.

8.2.11.2.1.2 uhScanRateConv2_DeiMode_t

Signature

```
typedef enum _uhScanRateConv2_DeiMode_t {
    uhScanRateConv2_NotApplicable         = 0x00000001,
    uhScanRateConv2_LineRepeat            = 0x00000002,
    uhScanRateConv2_LineFilter            = 0x00000004,
    uhScanRateConv2_FieldInsert           = 0x00000008,
    uhScanRateConv2_MotionAdaptive        = 0x00000010,
    uhScanRateConv2_MotionCompensated     = 0x00000020,
    uhScanRateConv2_AutoMotionAdaptive    = 0x00000040,
    uhScanRateConv2_AutoMotionCompensated = 0x00000080
} uhScanRateConv2_DeiMode_t, *puhScanRateConv2_DeiMode_t;
```

Qualifiers
enum-element.

Description
The re/de-interlacing modes.

Values

Table 1107

Name	Description
uhScanRateConv2_NotApplicable	De-interlacing is not applicable.
uhScanRateConv2_LineRepeat	De-interlacing by repeating the lines of a single field (consisting of only even or odd lines), filling the entire frame. This results in the video having effectively half the vertical resolution, scaled to the full resolution. This technique will cause a noticeable reduction in picture quality. This technique is often referred to as bob de-interlacing, because the fields are bobbed up and down..
uhScanRateConv2_LineFiltered	De-interlacing by applying a filter which uses multiple lines of a single field (consisting of only even or odd lines).
uhScanRateConv2_FieldInsert	De-interlacing by combining two consecutive fields into one progressive frame. When there is no motion between the consecutive fields this is a very good de-interlacing method. When there is motion, this method will introduce major motion artifacts. This technique is often referred to as weave de-interlacing, because the fields are simply weaved together.
uhScanRateConv2_MotionAdaptive	De-interlacing by combining the best of the line repeat uhScanRateConv2_LineRepeat and field insert method by use of a median filter. This filter approximates the field insert method when there is no motion and the line repeat when there is. To reduce detail flicker, vertical averaging is added

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhScanRateConv2_MotionCompensated	Basically this de-interlacing method is same as uhScanRateConv2_MotionAdaptive. However additionally motion information is used to generate the missing lines.
uhScanRateConv2_AutoMotionAdaptive	De-interlacing by either using the uhScanRateConv2_FieldInsert or uhScanRateConv2_MotionAdaptive method depending on the detected input source material (video or film). For video material, a median filter is used and for film material field insertion.
uhScanRateConv2_AutoMotionCompensated	De-interlacing by either using the uhScanRateConv2_FieldInsert or uhScanRateConv2_MotionCompensated method depending on the detected input source material (video or film). For video material, motion compensated is used and for film material field insertion.

Remarks
— None.

8.2.11.2.1.3 uhScanRateConv2_DeiModeSet_t

Signature
```
typedef Int32 uhScanRateConv2_DeiModeSet_t, *puhScanRateConv2_DeiModeSet_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of re/de-interlacing modes that are typically constructed by logical OR-ing of values of type `uhScanRateConv2_DeiMode_t`.

Constraints
— None.

8.2.11.2.1.4 uhScanRateConv2_McTcMode_t

Signature
```
typedef enum _uhScanRateConv2_McTcMode_t {
    uhScanRateConv2_McTcEnabled  = 0x00000001,
    uhScanRateConv2_McTcDisabled = 0x00000002
} uhScanRateConv2_McTcMode_t, *puhScanRateConv2_McTcMode_t;
```

Qualifiers
— enum-element

Description
The motion compensated temporal conversion modes.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1108

Name	Description
uhScanRateConv2_McTcEnabled	Motion compensated temporal conversion is enabled.
uhScanRateConv2_McTcDisabled	Motion compensated temporal conversion is enabled.

8.2.11.2.1.5 uhScanRateConv2_McTcModeSet_t

Signature

```
typedef Int32 uhScanRateConv2_McTcModeSet_t, *puhScanRateConv2_McTcModeSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of motion compensated temporal conversion modes that are typically constructed by logical OR-ing of values of type uhScanRateConv2_McTcMode_t.

8.2.11.2.1.6 uhScanRateConv2_Notification_t

Signature

```
typedef enum _uhScanRateConv2_Notification_t {
    uhScanRateConv2_OnSubscriptionChanged         = 0x00000001,
    uhScanRateConv2_OnAvailableAlgorithmsChanged  = 0x00000002,
    uhScanRateConv2_OnActualAlgorithmChanged      = 0x00000004
} uhScanRateConv2_Notification_t, *puhScanRateConv2_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhIScanRateConv2Ntf.

Values

Table 1109

Name	Description
uhScanRateConv2_OnSubscriptionChanged	Represents OnSubscriptionChanged.
uhScanRateConv2_OnAvailableAlgorithmsChanged	Represents OnAvailableAlgorithmsChanged.
uhScanRateConv2_OnActualAlgorithmChanged	Represents OnActualAlgorithmChanged.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11.2.1.7 uhScanRateConv2_NotificationSet_t

Signature

```
typedef Int32 uhScanRateConv2_NotificationSet_t, *puhScanRateConv2_NotificationSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions, which are typically constructed by logical OR-ing of values of type. These values are used in the *Subscribe* and *Unsubscribe* functions to specify the event notification to be enabled or disabled.

8.2.11.2.1.8 uhScanRateConv2_AllNotifications

Signature

```
const uhScanRateConv2_NotificationSet_t uhScanRateConv2_AllNotifications = 0x00000007;
```

Qualifiers

— None.

Description

Defines the set of all values of type. This constant can be used in calls of *Subscribe* and *Unsubscribe* functions to enable or disable all event notifications, respectively.

8.2.11.2.2 Model Types & Constants

None

8.2.11.3 Logical Component

8.2.11.3.1 Interface-Role Model

Figure 132 depicts the interface-role model of the Scan Rate Converter LC. It shows the roles and interfaces the Scan Rate Converter LC and its clients can have.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

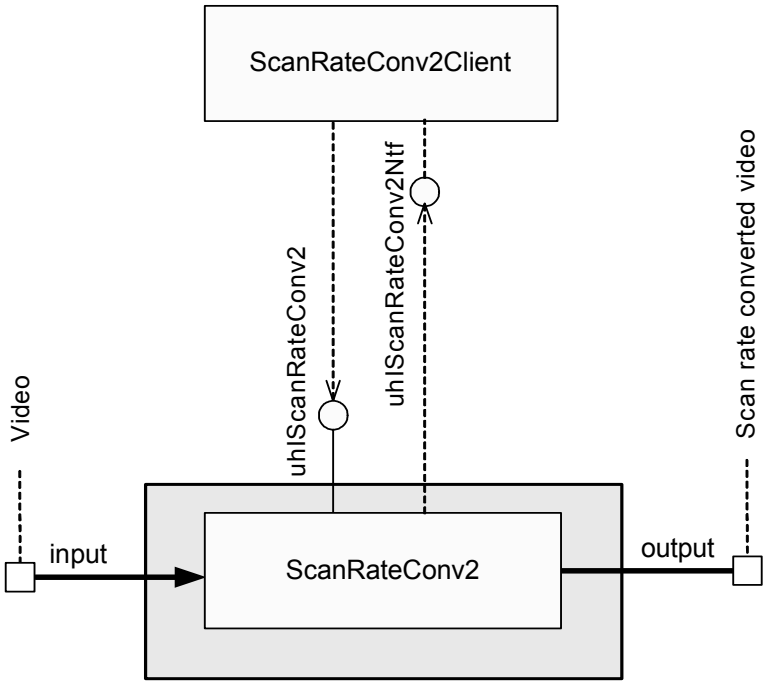


Figure 132 — Scan Rate Converter Interface-Role Model

The Scan Rate Converter LC (grey box) consists of the ScanRateConv role. This role provides one interface: `uhIScanRateConv2`. The `uhIScanRateConv2` interface controls the scan rate conversion mode. This interface is provided to the client `ScanRateConv2Client`. This client can optionally provide one callback interface: `uhIScanRateConv2Ntf`. This allows the client to subscribe to notifications of events in the Scan Rate Converter.

8.2.11.3.2 Diversity

8.2.11.3.2.1 Provided Interfaces

Table 1110

Role	Interface	Presence
`ScanrateConv`	`uhIScanRateConv2`	`mandatory`
`ScanRateConv2Client`	`uhIScanRateConv2Ntf`	`optional`

8.2.11.3.2.2 Configurable Items

Table 1111

Role	Attribute
`ScanRateConv`	`defaultAlgoList`

In case the preferred conversion algorithm is not available at current system constrains, the scanrate converter will use the next best algorithm in line as described in the `defaultAlgoList` as default. This way the client can provide design-time policy via this configurable item, in combination with end-user preference via interface.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The `defaultAlgoList` represents an ordered copy of the supported de-interlacing and temporal conversion modes.

8.2.11.3.2.3 Constraints

None.

8.2.11.3.3 Instantiation

8.2.11.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1112

Type	Object	Multiplicity
`ScanRateConv`	`conv`	1

8.2.11.3.3.2 Initial State

- `conv.preferredDeiMode == uhScanRateConv2_AutoMotionCompensated`
- `conv.preferredMcTcMode == uhScanRateConv2_McTcEnabled`

The following attributes are undefined:

- `conv.availableDeiModes`
- `conv.availableMcTcModes`
- `conv.supportedDeiModes`
- `conv.supportedMcTcModes`
- `conv.actualDeiMode`
- `conv.actualMcTcMode`

8.2.11.3.4 Execution Constraints

Per interface, at most one thread may have access to each interface. The only exception is thread-safe functions.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11.4 Roles

8.2.11.4.1 ScanrateConv

Signature

```
role ScanRateConv {
    uhScanRateConv2_DeiModeSet_t    availableDeiModes;
    uhScanRateConv2_McTcModeSet_t   availableMcTcModes;
    uhScanRateConv2_DeiModeSet_t    supportedDeiModes;
    uhScanRateConv2_McTcModeSet_t   supportedMcTcModes;
    uhScanRateConv2_DeiMode_t       preferredDeiMode;
    uhScanRateConv2_McTcMode_t      preferredMcTcMode;
    uhScanRateConv2_DeiMode_t       actualDeiMode;
    uhScanRateConv2_McTcMode_t      actualMcTcMode;
}
```

Qualifiers

— root

Description

The `ScanRateConv` controls which algorithms will be used for converting the video stream.

Independent Attributes

Table 1113

Name	Description
`pSupportedDeiModes`	Set of de-interlacing algorithms supported by current platform.
`pSupportedMcTcMode`	Set of temporal conversion modes supported by current platform

Dependent Attributes

Table 1114

Name	Description
pAvailableDeiModes	Set of de-interlacing algorithms available at current limitations/restrictions
pAvailableMcTcModes	Set of temporal conversion modes available at current limitations/restrictions.
preferredDeiMode	Preferred de-interlacing algorithm as set by client.
preferredMcTcMode	Preferred temporal conversion mode as set by client.
actualDeiMode	The actual de-interlacing algorithm that is used by the scan rate converter.
actualMcTcMode	The actual temporal conversion mode that is used by the scan rate converter.

Invariants

`None.`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Instantiation
A `ScanRateConv` is always instantiated as part of a Scan Rate Converter; see subclause 8.2.11.3.3.

Streaming Behavior
The `ScanRateConv` converts the scan rate of the video stream at the input into the requested output scan rate using the actual algorithm. In most cases, the actual algorithm is equal to the preferred algorithm for the current input format as set by the client. When the preferred algorithm is temporally unavailable because of system limitations, then the next best as described in the `defaultAlgoList` will become the actual algorithm.

Active Behavior
The active behavior of a subject conceptually consists of two activities:

— Executing pending subscribe/unsubscribe actions, i.e. actions that are a consequence of in-calls to the asynchronous `Subscribe` and `Unsubscribe` functions. No execution order is specified for these actions except for the subscribe/unsubscribe actions that relate to the same subscription (observer-cookie pair): they are serialized and executed in first-in-first-out order, i.e. in the order the `Subscribe`/`Unsubscribe` functions were called.

— Executing asynchronous (i.e. potentially delayed) calls to the `OnEvent` notification functions (including `OnSubscriptionChanged`) in reaction to the occurrence of events. No execution order is specified for these calls except for the calls that relate to the same subscription (observer-cookie pair): they are serialized and executed in first-in-first-out order, i.e. in the order of detecting the events.

The active behavior associated with the subscribe/unsubscribe actions is defined by the specifications of the asynchronous `Subscribe` and `Unsubscribe` functions. The active behavior associated with the calls to notification functions is specified in the event-action table below. The implicit assumption in all this is that the order of executing the subscribe/unsubscribe actions and the notifications is consistent with the two first-in-first-out rules described above.

Table 1115

Event	Action
Changing constraints / limitations / dependencies causing change in the availability of the conversion algorithms. For example the change of the preferred in- or output format.	— For each observer x and cookie c such that `uhScanRateConv2_OnAvailableAlgorithmsChanged` in `subscription[x,c]` the subject performs the following asynchronous function call: `OnAvailableAlgorithmsChanged()` — Modify `conv->pAvailableAlgorithms`
Changing constraints / limitations / dependencies causing the current conversion algorithm to be no longer available.	— For each observer x and cookie c such that `uhScanRateConv2_OnActualAlgorithmChanged` in `subscription[x,c]` the subject performs the following asynchronous function call: `OnActualAlgorithmChanged()` — Modify `conv.actualAlgorithm` with consecutive value of `defaultAlgoList`

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11.4.2 ScanRateConv2Client

Signature
```
role ScanRateConv2Client {}
```

Qualifiers
— actor

Description
A `ScanRateConv2Client` represents a user of the interface `uhIScanRateConv2`.

8.2.11.5 Interfaces

8.2.11.5.1 uhIScanRateConv2

Qualifiers
— root

Description
This interface controls the algorithm that is used to convert the scan rate of a video stream.

Interface ID
```
uuid( A0D524B7-0999-42a7-BD60-3A45CECDF571 )
```

Remarks
In the following function specifications, the name `conv` will be used to refer to the provider of this interface; see subclause 8.2.11.3.3.1.

8.2.11.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe(
    [in] uhIScanRateConv2Ntf              *pINotify,
    [in] UInt32                            cookie,
    [in] uhScanRateConv2_NotificationSet_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe(
    [in] uhIScanRateConv2Ntf              *pINotify,
    [in] UInt32                            cookie,
    [in] uhScanRateConv2_NotificationSet_t notifs );
```

Qualifiers

— unsubscribe-function

8.2.11.5.1.3 GetSupportedAlgorithms

Signature

```
uhErrorCode_t GetSupportedAlgorithms(
    [out] puhScanRateConv2_DeiModeSet_t  pSuppDeiModes,
    [out] puhScanRateConv2_McTcModeSet_t pSuppMcTcModes);
```

Qualifiers

— synchronous

Description

Returns orthogonal list supported de-interlacing and temporal conversion modes.

Parameters

Table 1116

Name	Description
`pSuppDeiModes`	Output parameter returning a list of supported de-interlacing modes.
`pSuppMcTcModes`	Output parameter returning a list of supported temporal conversion modes.

Return Values

Standard.

Precondition

`True`

Action

— None.

Postcondition

— `*pSuppDeiModes  == conv.supportedDeiModes`
— `*pSuppMcTcModes == conv.supportedDeiModes`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11.5.1.4 GetAvailableAlgorithms

Signature

```
uhErrorCode_t GetAvailableAlgorithms(
    [out] puhScanRateConv2_DeiModeSet_t  pAvailDeiModes,
    [out] puhScanRateConv2_McTcModeSet_t pAvailMcTcModes );
```

Qualifiers

— synchronous

Description

Returns orthogonal list available de-interlacing and temporal conversion modes.

Parameters

Table 1117

Name	Description
`pAvailDeiModes`	Output parameter returning a list of currently available de-interlacing modes.
`pAvailMcTcModes`	Output parameter returning a list of currently available temporal conversion modes.

Return Values

Standard.

Precondition

`True`

Action

— None.

Postcondition

— `*pAvailDeiModes  == conv.availableDeiModes`
— `*pAvailMcTcModes == conv.availableMcTcModes`

8.2.11.5.1.5 SetPreferredAlgorithm

Signature

```
uhErrorCode_t SetPreferredAlgorithm(
    [in] uhScanRateConv2_DeiMode_t  deiMode,
    [in] uhScanRateConv2_McTcMode_t mcTcMode);
```

Qualifiers

— synchronous

Description

Sets the preferred algorithm.When algoritm is not part of the list of available algorithms, the scan rate converter autonomously will select a default conversion algorithm as described by the configurable item `defaultAlgoList`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1118

Name	Description
`deiMode`	The preferred de-interlacing mode
`mcTcMode`	The preferred temporal conversion mode

Return Values
Standard.

Precondition
— `deiMode` in `conv.supportedDeiModes`
— `mcTcMode` in `conv.supportedMcTcModes`

Action
None

Postcondition
— `conv.preferredDeiMode == deiMode`
— `conv.preferredMcTcMode == mcTcMode`

8.2.11.5.1.6 GetPreferredAlgorithm

Signature

```
uhErrorCode_t GetPreferredAlgorithm(
    [out] puhScanRateConv2_DeiMode_t  pDeiMode,
    [out] puhScanRateConv2_McTcMode_t pMcTcMode);
```

Qualifiers
— synchronous

Description
Returns the preferred algorithm.

Parameters

Table 1119

Name	**Description**
`pDeiMode`	Output parameter returning the preferred de-interlacing mode
`pMcTcMode`	Output parameter returning the preferred temporal conversion mode

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— None

Postcondition
— `*pDeiMode  == conv.preferredDeiMode`
— `*pMcTcMode == conv.preferredMcTcMode`

8.2.11.5.1.7 GetActualAlgorithm

Signature
```
uhErrorCode_t GetActualAlgorithm(
    [out] puhScanRateConv2_DeiMode_t  pDeiMode,
    [out] puhScanRateConv2_McTcMode_t pMcTcMode);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the actual used algorithm.

Parameters

Table 1120

Name	Description
`pDeiMode`	Output parameter returning the actual used de-interlacing mode.
`pMcTcMode`	Output parameter returning the actual used temporal conversion mode.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pDeiMode        == conv.ActualDeiMode`
— `*pAvailMcTcModes == conv.actualMcTcMode`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.11.5.2 uhIScanRateConv2Ntf

Qualifiers
— callback

Description
This is the notification interface to be provided by Scan Rate Converter clients that wish to subscribe to notifications of Scan Rate Converter events.

Interface ID
```
uuid( 2C94B293-E7C1-4f27-95CE-544F9AB26B48 )
```

8.2.11.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                          cookie,
    [in] uhScanRateConv2_NotificationSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.11.5.2.2 OnAvailableAlgorithmsChanged

Signature
```
[async] Void OnAvailableAlgorithmsChanged(
    [in] UInt32                       cookie,
    [in] uhScanRateConv2_DeiModeSet_t   availDeiModes,
    [in] uhScanRateConv2_McTcModeSet_t  availMcTcModes);
```

Qualifiers
None.

Description
This function is used by a `ScanRateConv` to notify a client that list of allowed algorithms has changed.

Parameters

Table 1121

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`availDeiModes`	Output parameter returning a list of currently available de-interlacing modes.
`availMcTcModes`	Output parameter returning a list of currently available temporal conversion modes.

Return Values
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
— any callback-compliant action.

Postcondition
`True`

8.2.11.5.2.3 OnActualAlgorithmChanged

Signature
```
[async] Void OnActualAlgorithmChanged(
    [in] UInt32                     cookie,
    [in] uhScanRateConv2_DeiMode_t  deiMode,
    [in] uhScanRateConv2_McTcMode_t mcTcMode);
```

Qualifiers
— None.

Description
This function is used by a `ScanRateConv` to notify a client that the actual algorithm has changed.

Parameters

Table 1122

Name	Description
`cookie`	The value that was passed by the ScanRateConvClient at subscription time.
`deiMode`	Output parameter returning the actual used de-interlacing mode.
`mcTcMode`	Output parameter returning the actual used temporal conversion mode.

Return Values
None.

Precondition
`True`

Action
— any callback-compliant action

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.12 UhlSharpEnh

8.2.12.1 Concepts

8.2.12.1.1 Sharpness Enhancement

The Sharpness Enhancement logical component described in this document deals with enhancing the sharpness of the luminance (Y) component of a YUV signal. Via the main interface (`uhISharpEnh`) of the logical component, the client can set a sharpness enhancement (SE) level. A higher SE level corresponds with more sharpening.

The sharpness enhancement methods used are Luminance Transient Improvement (LTI) and peaking. LTI steepens transients (see Figure 133) and peaking adds undershoots and overshoots to transients (see Figure 134).

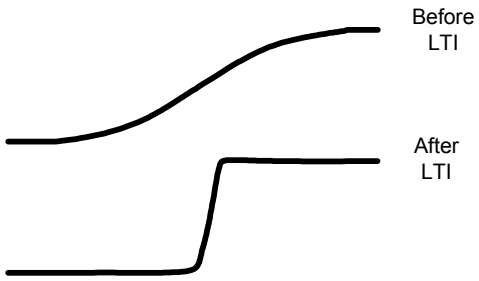


Figure 133 — LTI: steepening transients in the luminance signal.

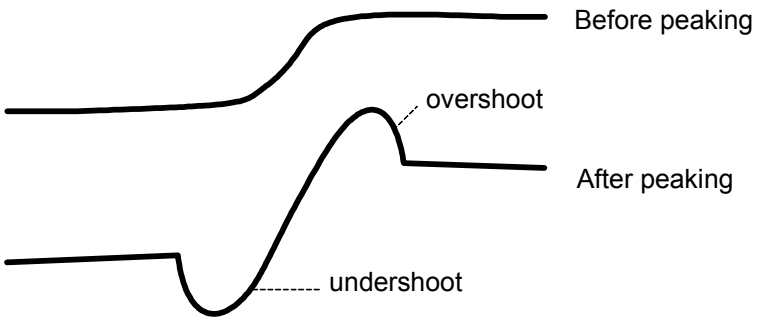


Figure 134 — Peaking: adding undershoots and overshoots to transients in the luminance signal.

8.2.12.1.2 Color Dependent Sharpness

The logical component optionally supports an interface (`uhISharpEnhCds`), via which another sharpness enhancement method can be controlled: Color Dependent Sharpness (CDS). CDS is a form of peaking applied to the luminance signal in red or magenta areas only. The client can set a CDS level (where a higher level corresponds with more sharpening).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.12.2 Types & Constants

8.2.12.2.1 Public Types & Constants

8.2.12.2.1.1 uhSharpEnh_Notification_t

Signature

```
typedef enum _uhSharpEnh_Notification_t {
    uhSharpEnh_SubscriptionChanged = 0x00000001,
    uhSharpEnh_LevelChanged        = 0x00000002
} uhSharpEnh_Notification_t, *puhSharpEnh_Notification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhISharpEnhNtf`.

Values

Table 1123

Name	Description
`uhSharpEnh_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhSharpEnh_LevelChanged`	Represents `OnLevelChanged`.

8.2.12.2.1.2 uhSharpEnh_Notifications_t

Signature

```
typedef Int32 uhSharpEnh_Notifications_t,
              *puhSharpEnh_Notifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`SharpEnh_Notification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions of the interface `uhISharpEnh` to specify the event notifications to be enabled or disabled.

8.2.12.2.1.3 uhSharpEnh_AllNotifications

Signature

```
const uhSharpEnh_Notifications_t
    uhSharpEnh_AllNotifications = 0x00000003;
```

Qualifiers

— None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set of all values of type uh`SharpEnh_Notification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions of the interface `uhISharpEnh` to enable or disable all event notifications, respectively.

8.2.12.2.1.4 uhSharpEnh_CdsNotification_t

Signature
```
typedef enum _uhSharpEnh_CdsNotification_t {
    uhSharpEnh_CdsSubscriptionChanged = 0x00000001,
    uhSharpEnh_CdsLevelChanged        = 0x00000002
} uhSharpEnh_CdsNotification_t, *puhSharpEnh_CdsNotification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhISharpEnhCdsNtf`.

Values

Table 1124

Name	Description
`uhSharpEnh_CdsSubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhSharpEnh_CdsLevelChanged`	Represents `OnLevelChanged`.

8.2.12.2.1.5 uhSharpEnh_CdsNotifications_t

Signature
```
typedef Int32 uhSharpEnh_CdsNotifications_t,
              *puhSharpEnh_CdsNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`SharpEnh_CdsNotification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions of the interface `uhISharpEnhCds` to specify the event notifications to be enabled or disabled.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.12.2.1.6 uhSharpEnh_CdsAllNotifications

Signature

```
const uhSharpEnh_CdsNotifications_t
    uhSharpEnh_CdsAllNotifications = 0x00000003;
```

Qualifiers

— None.

Description

Defines the set of all values of type uh`SharpEnh_CdsNotification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions of the interface `uhISharpEnhCds` to enable or disable all event notifications, respectively.

8.2.12.2.2 Model Types & Constants

None.

8.2.12.3 Logical Component

8.2.12.3.1 Interface-Role Model

The interface role-model is depicted in Figure 135.

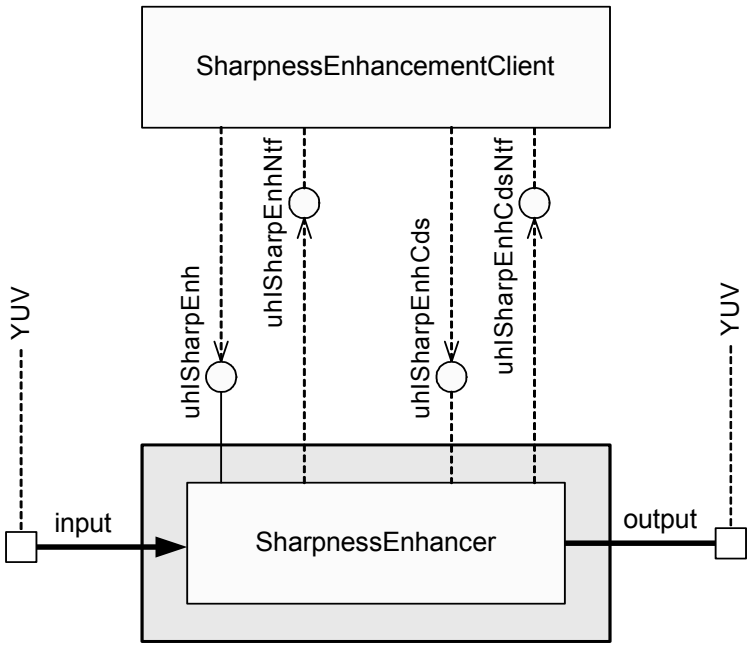


Figure 135 — Interface-Role Model of the Sharpness Enhancement logical component

The Sharpness Enhancement logical component (grey box) consists of the `SharpnessEnhancer` role. This role provides the interface `uhISharpEnh` for controlling the level of SE and the interface `uhISharpEnhCds` for controlling the level of CDS. The control interfaces are meant for use by the client (`SharpnessEnhancementClient`). This client can optionally provide call-back interfaces

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(uhISharpEnhNtf and uhISharpEnhCdsNtf) allowing it to subscribe to notifications of events in the Sharpness Enhancement logical component.

8.2.12.3.2 Diversity

8.2.12.3.2.1 Provided Interfaces

Table 1125

Role	Interface	Presence
SharpnessEnhancer	uhISharpEnh	mandatory
SharpnessEnhancementClient	uhISharpEnhNtf	optional
SharpnessEnhancer	uhISharpEnhCds	optional
SharpnessEnhancementClient	uhISharpEnhCdsNtf	optional

8.2.12.3.2.2 Configurable Items

Table 1126

Role	Attribute
SharpnessEnhancer	minSharpnessLevel
SharpnessEnhancer	maxSharpnessLevel
SharpnessEnhancer	minCdsLevel
SharpnessEnhancer	maxCdsLevel

8.2.12.3.2.3 Constraints

— maxSharpnessLevel >= *minSharpnessLevel*
— maxCdsLevel >= *minCdsLevel*

8.2.12.3.3 Instantiation

8.2.12.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1127

Type	Object	Multiplicity
SharpnessEnhancer	se	1

8.2.12.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— se.sharpnessEnabled == False
— se.sharpnessLevel == se.minSharpnessLevel
— se.cdsEnabled == False

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— se.cdsLevel == se.minCdsLevel

8.2.12.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.2.12.4 Roles

8.2.12.4.1 SharpnessEnhancer

Signature

```
role SharpnessEnhancer {
   Bool                   sharpnessEnabled;
   const UInt32        minSharpnessLevel;
   const UInt32        maxSharpnessLevel;
   UInt32              sharpnessLevel;
   Bool                   cdsEnabled;
   const UInt32        minCdsLevel;
   const UInt32        maxCdsLevel;
   UInt32              cdsLevel;
}
```

Qualifiers

— root

Description

The `SharpnessEnhancer` role controls the level for enhancing the sharpness of a luminance signal via LTI and peaking, and it controls the level for enhancing the sharpness of a luminance signal via CDS. The levels of SE and CDS can be set and both sharpness enhancement methods of the `SharpnessEnhancer` can be enabled or disabled.

Independent Attributes

Table 1128

Name	Description
`sharpnessEnabled`	Indicates whether SE is enabled or not.
`minSharpnessLevel`	The minimum SE level.
`maxSharpnessLevel`	The maximum SE level.
`sharpnessLevel`	The current SE level.
`cdsEnabled`	Indicates whether CDS is enabled or not.
`minCdsLevel`	The minimum CDS level.
`maxCdsLevel`	The maximum CDS level.
`cdsLevel`	The current CDS level.

Dependent Attributes

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Invariants
— minSharpnessLevel <= *sharpnessLevel* <= maxSharpnessLevel
— *minCdsLevel* <= cdsLevel <= *maxCdsLevel*

Instantiation
See subclause 8.2.12.3.3.

Streaming Behavior
The `SharpnessEnhancer` has a YUV signal as an input. When SE and/or CDS are enabled, the `SharpnessEnhancer` will perfom SE and/or CDS on the input stream. Enhancements only apply to the Y component of the signal (CDS uses the UV signal as input to enhance the sharpness of the Y signal). The processed stream is passed to the output.

Active Behavior
The `SharpnessEnhancer` has standard notification behavior (see `uhISharpEnhNtf` and `uhISharpEnhCdsNtf`).

8.2.12.4.2 SharpnessEnhancementClient

Signature
```
role SharpnessEnhancementClient {}
```

Qualifiers
— actor

Description
A `SharpnessEnhancementClient` represents a user of the interfaces `uhISharpEnh` and `uhISharpEnhCds`.

8.2.12.5 Interfaces

8.2.12.5.1 uhISharpEnh

Qualifiers
None.

Description
This interface controls the level of SE that is applied to the video signal.

Interface ID
```
uuid( D539C012-4D67-42f9-980F-2BFF7260D93E )
```

Remarks
In the following function specifications, the name se will be used to refer to the provider of this interface; see subclause 8.2.12.3.3.1.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.12.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhISharpEnhNtf               *pINotify,
    [in] UInt32                       cookie,
    [in] uhSharpEnh_Notifications_t   notifs );
```

Qualifiers

— subscribe-function

8.2.12.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhISharpEnhNtf               *pINotify,
    [in] UInt32                       cookie,
    [in] uhSharpEnh_Notifications_t   notifs );
```

Qualifiers

— unsubscribe-function

8.2.12.5.1.3 GetEnabled

Signature

```
uhErrorCode_t GetEnabled (
    [out] pBool       pEnabled);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether SE is enabled or not.

Parameters

Table 1129

Name	Description
`pEnabled`	Output parameter returning whether SE is enabled (`True`) or disabled (`False`).

Return Values

Standard.

Precondition

`True`

Action

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `*pEnabled == se.sharpnessEnabled`

8.2.12.5.1.4 Enable

Signature
```
uhErrorCode_t Enable (
    [in] Bool        enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables SE.

Parameters

Table 1130

Name	Description
`enable`	Boolean to enable (`True`) or disable (`False`) SE.

Return Values
Standard.

Precondition
`True`

Action
— Modify se.sharpnessEnabled

Postcondition
`se.sharpnessEnabled == enable`

8.2.12.5.1.5 GetLevel

Signature
```
uhErrorCode_t GetLevel (
    [out] pUInt32             pLevel );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current SE level.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1131

Name	Description
`pLevel`	Output parameter returning the current SE level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— *`*pLevel == se.sharpnessLevel`*

8.2.12.5.1.6 SetLevel

Signature
```
uhErrorCode_t SetLevel (
    [in] UInt32             level );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the SE level.

Parameters

Table 1132

Name	Description
`level`	The new SE level.

Return Values
Standard.

Precondition
— `se.minSharpnessLevel <=` level `<= se.maxSharpnessLevel`

Action
— Modify `se.sharpnessLevel`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `se.sharpnessLevel == level`

8.2.12.5.1.7 GetLevelRange

Signature
```
uhErrorCode_t GetLevelRange (
    [out] pUInt32       pMinLevel,
    [out] pUInt32       pMaxLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum SE levels.

Parameters

Table 1133

Name	Description
`pMinLevel`	Output parameter returning the minimum SE level.
`pMaxLevel`	Output parameter returning the maximum SE level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pMinLevel == se.minSharpnessLevel`
— `*pMinLevel == se.maxSharpnessLevel`

8.2.12.5.2 uhISharpEnhNtf

Qualifiers
— callback

Description
This is a notification interface to be provided by clients of the Sharpness Enhancement logical component that wish to subscribe to notifications of SE related events of the Sharpness Enhancement logical component.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
uuid(1C1613E9-7A82-4f59-A90E-6DAA5728A1EE)

8.2.12.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                        cookie,
    [in] uhSharpEnh_Notifications_t    notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.12.5.2.2 OnLevelChanged

Signature
```
[async] Void OnLevelChanged (
    [in] UInt32      cookie,
    [in] UInt32      level);
```

Qualifiers
None.

Description
This function is called by the `SharpnessEnhancer` whenever it detects a change of the SE level.

Parameters

Table 1134

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`level`	The new SE level.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.12.5.3 uhISharpEnhCds

Qualifiers
None.

Description
This interface controls the level of CDS that is applied to the luminance signal.

Interface ID
`uuid( 15D68995-BD11-4047-A491-152D36AAEE0E )`

Remarks
In the following function specifications, the name se will be used to refer to the provider of this interface; see subclause 8.2.12.3.3.1.

8.2.12.5.3.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhISharpEnhCdsNtf              *pINotify,
    [in] UInt32                         cookie,
    [in] uhSharpEnh_CdsNotifications_t  notifs );
```

Qualifiers
— subscribe-function

8.2.12.5.3.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhISharpEnhCdsNtf              *pINotify,
    [in] UInt32                         cookie,
    [in] uhSharpEnh_CdsNotifications_t  notifs );
```

Qualifiers
— unsubscribe-function

8.2.12.5.3.3 GetEnabled

Signature
```
uhErrorCode_t GetEnabled (
    [out] pBool       pEnabled);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether CDS is enabled or not.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1135

Name	Description
pEnabled	Output parameter returning whether CDS is enabled (True) or disabled (False).

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pEnabled == se.cdsEnabled

8.2.12.5.3.4 Enable

Signature

```
uhErrorCode_t Enable (
    [in] Bool        enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables CDS.

Parameters

Table 1136

Name	Description
enable	Boolean to enable (True) or disable (False) CDS.

Return Values
Standard.

Precondition
True

Action
— Modify se.cdsEnabled

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— `se.cdsEnabled` == *enable*

8.2.12.5.3.5 GetLevel

Signature
```
uhErrorCode_t GetLevel (
    [out] pUInt32             pLevel );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current CDS level.

Parameters

Table 1137

Name	Description
`pLevel`	Output parameter returning the current CDS level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pLevel == se.cdsLevel`

8.2.12.5.3.6 SetLevel

Signature
```
uhErrorCode_t SetLevel (
    [in] UInt32             level );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the CDS level.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1138

Name	Description
`level`	The new CDS level.

Return Values
Standard.

Precondition
— `se.minCdsLevel <= level <= se.maxCdsLevel`

Action
— Modify `se.cdsLevel`

Postcondition
— `se.cdsLevel` == level

8.2.12.5.3.7 GetLevelRange

Signature
```
uhErrorCode_t GetLevelRange (
    [out] pUInt32       pMinLevel,
    [out] pUInt32       pMaxLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum CDS levels.

Parameters

Table 1139

Name	Description
`pMinLevel`	Output parameter returning the minimum CDS level.
`pMaxLevel`	Output parameter returning the maximum CDS level.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— **pMinLevel* == se.minCdsLevel
— *pMaxLevel == se.maxCdsLevel

8.2.12.5.4 uhlSharpEnhCdsNtf

Qualifiers
— callback

Description
This is a notification interface to be provided by clients of the Sharpness Enhancement logical component that wish to subscribe to notifications of CDS related events of the Sharpness Enhancement logical component.

Interface ID
uuid(1B1EA294-F6F9-47e1-86D0-431A2E3B38DB)

8.2.12.5.4.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                          cookie,
    [in] uhSharpEnh_CdsNotifications_t   notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.12.5.4.2 OnLevelChanged

Signature
```
[async] Void OnLevelChanged (
    [in] UInt32      cookie,
    [in] UInt32      level );
```

Qualifiers
None.

Description
This function is called by the SharpnessEnhancer whenever it detects a change of the CDS level.

Parameters

Table 1140

Name	Description
cookie	The value that was passed by the client at subscription time.
level	The new CDS level.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.2.13 UhlSharpMeas

8.2.13.1 Concepts

8.2.13.1.1 Luma and color sharpness

The Sharpness Measurement logical component has a YUV signal as input. It measures the sharpness of the Y signal (*luma sharpness*) and the sharpness of the UV signal (*color sharpness*). The component provides an interface dealing with luma sharpness and one dealing with color sharpness.

8.2.13.1.2 Sharpness Measurement

The Sharpness Measurement logical component offers an abstract view on the sharpness of a video signal. A discrete range of sharpness "levels" is defined by *minLevel* and *maxLevel*. The current sharpness *level* of the measured signal is a number within that range, corresponding to the sharpness of the signal (Figure 136). The meter scale defined by this range is a linear scale, so going from one level to the next always corresponds with (more or less) the same increase in sharpness.

The extremes of the meter scale, *minLevel* and *maxLevel*, do not necessarily correspond with an absolute minimum and an absolute maximum sharpness, so when the meter has reached *maxLevel* and the signal becomes sharper, the client won't notice it it (the inverse holds for *minLevel*).

Luma and color sharpness measurement each have their own sharpness level range: [`lumaMinLevel..lumaMaxLevel`] and [`colorMinLevel.. colorMaxLevel`] respectively.

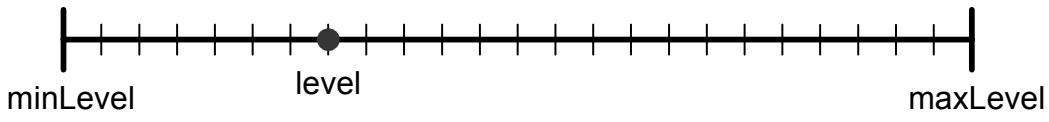


Figure 136 — A discrete linear meter scale.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.13.2 Types & Constants

8.2.13.2.1 Public Types & Constants

8.2.13.2.1.1 uhSharpMeas_ValStat_t

Signature

```
typedef enum _uhSharpMeas_ValStat_t {
    uhSharpMeas_ValStatInvalid = -2,
    uhSharpMeas_ValStatUnknown = -1,
    uhSharpMeas_ValStatKnown   =  1
} uhSharpMeas_ValStat_t, *puhSharpMeas_ValStat_t;
```

Qualifiers
None.

Description
This type represents the validity status of the related property/properties.

Values

Table 1141

Name	Description
uhSharpMeas_ValStatInvalid	The property/properties are temporarily not available.
uhSharpMeas_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhSharpMeas_ValStatKnown	The property/properties are available and determined.

8.2.13.2.1.2 uhSharpMeas_LumaNotification_t

Signature

```
typedef enum _uhSharpMeas_LumaNotification_t {
    uhSharpMeas_LumaSubscriptionChanged = 0x00000001,
    uhSharpMeas_LumaLevelChanged        = 0x00000002
} uhSharpMeas_LumaNotification_t, *puhSharpMeas_LumaNotification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhISharpMeasLumaNtf.

Values

Table 1142

Name	Description
uhSharpMeas_LumaSubscriptionChanged	Represents OnSubscriptionChanged.
uhSharpMeas_LumaLevelChanged	Represents OnLumaSharpnessLevelChanged.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.13.2.1.3 uhSharpMeas_LumaNotifications_t

Signature

```
typedef Int32 uhSharpMeas_LumaNotifications_t,
              *puhSharpMeas_LumaNotifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uh`SharpMeas_LumaNotification_t`. These values are used in the `Subscribe` and `Unsubscribe` functions of `uhISharpMeasLuma` to specify the event notifications to be enabled or disabled.

8.2.13.2.1.4 uhSharpMeas_LumaAllNotifications

Signature

```
const uhSharpMeas_LumaNotifications_t
    uhSharpMeas_LumaAllNotifications = 0x00000003;
```

Qualifiers

— None.

Description

Defines the set of all values of type uh`SharpMeas_LumaNotification_t`. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions of `uhISharpMeasLuma` to enable or disable all event notifications, respectively.

8.2.13.2.1.5 uhSharpMeas_ColorNotification_t

Signature

```
typedef enum _uhSharpMeas_ColorNotification_t {
    uhSharpMeas_ColorSubscriptionChanged = 0x00000001,
    uhSharpMeas_ColorLevelChanged        = 0x00000002
} uhSharpMeas_ColorNotification_t, *puhSharpMeas_ColorNotification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhISharpMeasColorNtf`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1143

Name	Description
uhSharpMeas_ColorSubscriptionChanged	Represents OnSubscriptionChanged.
uhSharpMeas_ColorLevelChanged	Represents OnColorSharpnessLevelChanged.

8.2.13.2.1.6 uhSharpMeas_ColorNotifications_t

Signature

```
typedef Int32 uhSharpMeas_ColorNotifications_t,
              *puhSharpMeas_ColorNotifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type uhSharpMeas_ColorNotification_t. These values are used in the Subscribe and Unsubscribe functions of uhISharpMeasColor to specify the event notifications to be enabled or disabled.

8.2.13.2.1.7 uhSharpMeas_ColorAllNotifications

Signature

```
const uhSharpMeas_ColorNotifications_t
      uhSharpMeas_ColorAllNotifications = 0x00000003;
```

Qualifiers

— None.

Description

Defines the set of all values of type uhSharpMeas_ColorNotification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions of uhISharpMeasColor to enable or disable all event notifications, respectively.

8.2.13.2.2 Model Types & Constants

None.

8.2.13.3 Logical Component

8.2.13.3.1 Interface-Role Model

The interface role-model is depicted in Figure 137

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

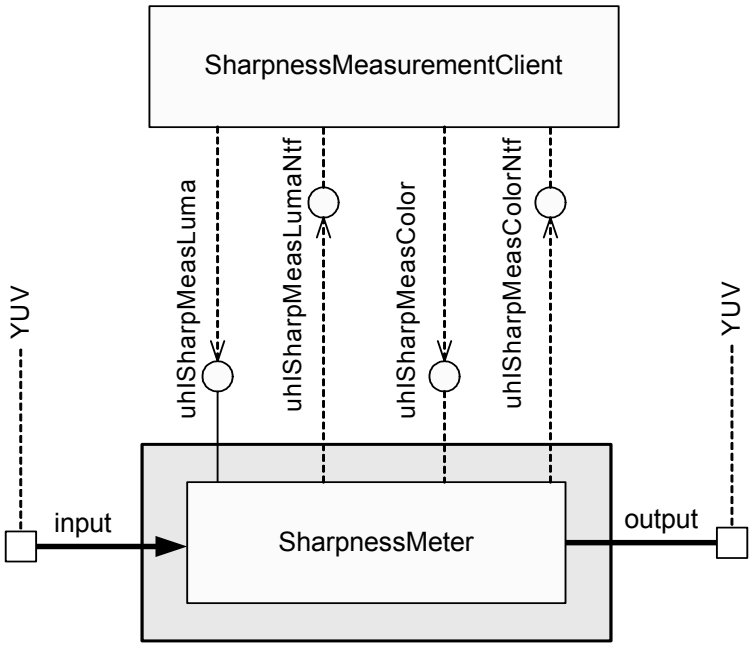


Figure 137 — Interface-Role Model of the Sharpness Measurement logical component

The Sharpness Measurement logical component (grey box) consists of the `SharpnessMeter` role. This role provides an interface (`uhISharpMeasLuma`) for retrieving the luma sharpness measurements and an interface (`uhISharpMeasColor`) for retrieving the color sharpness measurements. The control interfaces are meant for use by the client (`SharpnessMeasurementClient`). This client can optionally provide call-back interfaces (`uhISharpMeasLumaNtf` and/or `uhISharpMeasColorNtf`) allowing it to subscribe to notifications of events in the Sharpness Measurement logical component.

8.2.13.3.2 Diversity

8.2.13.3.2.1 Provided Interfaces

Table 1144

Role	Interface	Presence
`SharpnessMeter`	`uhISharpMeasLuma`	mandatory
`SharpnessMeter`	`uhISharpMeasColor`	optional
`SharpnessMeasurementClient`	`uhISharpMeasLumaNtf`	optional
`SharpnessMeasurementClient`	`uhISharpMeasColorNtf`	optional

8.2.13.3.2.2 Configurable Items

Table 1145

Role	Attribute
`SharpnessMeter`	`lumaMinLevel`
`SharpnessMeter`	`lumaMaxLevel`
`SharpnessMeter`	`colorMinLevel`
`SharpnessMeter`	`colorMaxLevel`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.13.3.2.3 Constraints

— lumaMaxLevel >= *lumaMinLevel*
— colorMaxLevel >= *colorMinLevel*

8.2.13.3.3 Instantiation

8.2.13.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1146

Type	Object	Multiplicity
`SharpnessMeter`	`shm`	1

8.2.13.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
— shm.lumaStatus == uhSharpMeas_ValStatUnknown
— shm.lumaEnabled == False
— shm.lumaLevel   == undefined
— shm.colorStatus == uhSharpMeas_ValStatUnknown
— shm.colorEnabled == False
— shm.colorLevel   == undefined
```

8.2.13.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.13.4 Roles

8.2.13.4.1 SharpnessMeter

Signature

```
role SharpnessMeter {
    uhSharpMeas_ValStat_t lumaStatus;
    Bool                  lumaEnabled;
    const UInt32          lumaMinLevel;
    const UInt32          lumaMaxLevel;
    UInt32                lumaLevel;
    uhSharpMeas_ValStat_t colorStatus;
    Bool                  colorEnabled;
    const UInt32          colorMinLevel;
    const UInt32          colorMaxLevel;
    UInt32                colorLevel;
}
```

Qualifiers

— root

Description

The `SharpnessMeter` role provides means to get an indication of the luma and color sharpness of a video signal. The luma and color sharpness levels can be queried and each of the two types of measurement of the `SharpnessMeter` can be enabled or disabled.

Independent Attributes

Table 1147

Name	Description
`lumaStatus`	Validity state of the luma sharpness level property.
`lumaEnabled`	Indicates whether luma sharpness measurement is enabled or not.
`lumaMinLevel`	Minimum luma sharpness level.
`lumaMaxLevel`	Maximum luma sharpness level.
`lumaLevel`	Current luma sharpness level.
`colorStatus`	Validity state of the color sharpness level property.
`colorEnabled`	Indicates whether color sharpness measurement is enabled or not.
`colorMinLevel`	Minimum color sharpness level.
`colorMaxLevel`	Maximum color sharpness level.
`colorLevel`	Current color sharpness level.

Dependent Attributes

None.

Invariants

— `lumaMinLevel <= lumaLevel <= lumaMaxLevel`
— `colorMinLevel <= colorLevel <= colorMaxLevel`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Instantiation
See subclause 8.2.13.3.3.

Streaming Behavior
The `SharpnessMeter` has a YUV signal as an input. When the luma sharpness measurement of SharpnessMeter is enabled, it will perform sharpness measurement on the Y signal and when color sharpness measurement is enabled, it will perform sharpness measurement on the UV signal.

Active Behavior
The `SharpnessMeter` has standard notification behavior (see `uhISharpMeasLumaNtf` and `uhISharpMeasColorNtf`).

8.2.13.4.2 SharpnessMeasurementClient

Signature
```
role SharpnessMeasurementClient {}
```

Qualifiers
— actor

Description
A `SharpnessMeasurementClient` represents a user of the `uhISharpMeasLuma` and `uhISharpMeasColor` interfaces.

8.2.13.5 Interfaces

8.2.13.5.1 uhISharpMeasLuma

Qualifiers
None.

Description
This interface provides means to get an indication of the luma sharpness of the video signal.

Interface ID
`uuid( CE72C910-3B17-4864-8010-22CF2E873294 )`

Remarks
In the following function specifications, the name `shm` will be used to refer to the provider of this interface; see subclause 8.2.13.3.3.1.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.13.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhISharpMeasLumaNtf              *pINotify,
    [in] UInt32                           cookie,
    [in] uhSharpMeas_LumaNotifications_t  notifs );
```

Qualifiers

— subscribe-function

8.2.13.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhISharpMeasLumaNtf              *pINotify,
    [in] UInt32                           cookie,
    [in] uhSharpMeas_LumaNotifications_t  notifs );
```

Qualifiers

— unsubscribe-function

8.2.13.5.1.3 GetEnabled

Signature

```
uhErrorCode_t GetEnabled (
    [out] pBool       pEnabled);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether luma sharpness measurement is enabled or not.

Parameters

Table 1148

Name	Description
`pEnabled`	Output parameter returning whether luma sharpness measurement is enabled (`True`) or disabled (`False`).

Return Values

Standard.

Precondition

`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— `*pEnabled == shm.lumaEnabled`

8.2.13.5.1.4 Enable

Signature
```
uhErrorCode_t Enable (
    [in] Bool        enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables luma sharpness measurement.

Parameters

Table 1149

Name	Description
`enable`	Boolean to enable (`True`) or disable (`False`) luma sharpness measurement.

Return Values
Standard.

Precondition
`True`

Action
— Modify shm.lumaEnabled

Postcondition
— `shm.lumaEnabled == enable`

8.2.13.5.1.5 GetLevel

Signature
```
uhErrorCode_t GetLevel (
    [out] puhSharpMeas_ValStat_t pStatus,
    [out] pUInt32                pLevel );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the current luma sharpness level.

Parameters

Table 1150

Name	Description
`pStatus`	Output parameter that indicates the validity of the luma sharpness level parameter.
`pLevel`	Output parameter returning the current luma sharpness level.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
```
*pStatus == shm.lumaStatus
*pLevel == shm.lumaLevel
```

8.2.13.5.1.6 GetLevelRange

Signature
```
uhErrorCode_t GetLevelRange (
    [out] pUInt32       pMinLevel,
    [out] pUInt32       pMaxLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum luma sharpness levels.

Parameters

Table 1151

Name	Description
`pMinLevel`	Output parameter returning the minimum luma sharpness level.
`pMaxLevel`	Output parameter returning the maximum luma sharpness level.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
— `*`pMinLevel `== shm.lumaMinLevel`
— `*`*pMaxLevel* `== shm.lumaMaxLevel`

8.2.13.5.2 uhlSharpMeasLumaNtf

Qualifiers
— callback

Description
This is a notification interface to be provided by clients of the Sharpness Measurement logical component that wish to subscribe to notifications of luma sharpness related events of the Sharpness Measurement logical component.

Interface ID
`uuid( F1C25759-F138-4463-AF6D-D519B51D187E )`

8.2.13.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                          cookie,
    [in] uhSharpMeas_LumaNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.13.5.2.2 OnLumaSharpnessLevelChanged

Signature
```
[async] Void OnLumaSharpnessLevelChanged (
    [in] UInt32                cookie,
    [in] uhSharpMeas_ValStat_t status,
    [in] UInt32                level);
```

Qualifiers
None.

Description
This function is called by the `SharpnessMeter` whenever it detects a change of the luma sharpness level.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1152

Name	Description
cookie	The value that was passed by the client at subscription time.
status	Parameter that indicates the validity of the luma sharpness level parameter.
level	The new luma sharpness level.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action.

Postcondition
True

8.2.13.5.3 uhISharpMeasColor

Qualifiers
None.

Description
This interface provides means to get an indication of the color sharpness of the video signal.

Interface ID
uuid(06153E0B-63E7-4f13-A79D-B2CD59CEC713)

Remarks
— In the following function specifications, the name shm will be used to refer to the provider of this interface; see subclause 8.2.13.3.3.1.

8.2.13.5.3.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhISharpMeasColorNtf              *pINotify,
    [in] UInt32                            cookie,
    [in] uhSharpMeas_ColorNotifications_t  notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.13.5.3.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhISharpMeasColorNtf               *pINotify,
    [in] UInt32                             cookie,
    [in] uhSharpMeas_ColorNotifications_t   notifs );
```

Qualifiers

— unsubscribe-function

8.2.13.5.3.3 GetEnabled

Signature

```
uhErrorCode_t GetEnabled (
    [out] pBool       pEnabled);
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether color sharpness measurement is enabled or not.

Parameters

Table 1153

Name	Description
`pEnabled`	Output parameter returning whether color sharpness measurement is enabled (`True`) or disabled (`False`).

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*`pEnabled `== shm.colorEnabled`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.13.5.3.4 Enable

Signature
```
uhErrorCode_t Enable (
    [in] Bool        enable );
```

Qualifiers
— synchronous

— single-threaded

Description
Enables or disables color sharpness measurement.

Parameters

Table 1154

Name	Description
`enable`	Boolean to enable (`True`) or disable (`False`) color sharpness measurement.

Return Values
Standard.

Precondition
`True`

Action
— Modify `shm.colorEnabled`

Postcondition
— `shm.colorEnabled` == *enable*

8.2.13.5.3.5 GetLevel

Signature
```
uhErrorCode_t GetLevel (
    [out] puhSharpMeas_ValStat_t pStatus,
    [out] pUInt32                pLevel );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the current color sharpness level.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1155

Name	Description
pStatus	Output parameter that indicates the validity of the color sharpness level parameter.
pLevel	Output parameter returning the current color sharpness level.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pStatus == shm.colorStatus
— *pLevel == shm.colorLevel

8.2.13.5.3.6 GetLevelRange

Signature

```
uhErrorCode_t GetLevelRange (
    [out] pUInt32       pMinLevel,
    [out] pUInt32       pMaxLevel);
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and the maximum color sharpness levels.

Parameters

Table 1156

Name	Description
pMinLevel	Output parameter returning the minimum color sharpness level.
pMaxLevel	Output parameter returning the maximum color sharpness level.

Return Values
Standard.

Precondition
True

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— **pMinLevel* == shm.colorMinLevel
— *pMaxLevel == shm.colorMaxLevel

8.2.13.5.4 uhISharpMeasColorNtf

Qualifiers
— callback

Description
This is a notification interface to be provided by clients of the Sharpness Measurement logical component that wish to subscribe to notifications of color sharpness related events of the Sharpness Measurement logical component.

Interface ID
uuid(DEE023F7-8223-4bea-AE13-C3F405A5D374)

8.2.13.5.4.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                          cookie,
    [in] uhSharpMeas_ColorNotifications_t   notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.13.5.4.2 OnColorSharpnessLevelChanged

Signature
```
[async] Void OnColorSharpnessLevelChanged (
    [in] UInt32                cookie,
    [in] uhSharpMeas_ValStat_t status,
    [in] UInt32                level);
```

Qualifiers
None.

Description
This function is called by the SharpnessMeter whenever it detects a change of the color sharpness level.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1157

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`status`	Parameter that indicates the validity of the color sharpness level parameter.
`level`	The new color sharpness level.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

8.2.14 UhlVfeat

8.2.14.1 Concepts

8.2.14.1.1 Basic Video Featuring

The function of a BVF is to improve the perceived quality of a video display. It aims to tune the basic video parameters brightness, contrast, saturation, hue, and white point to the end user preferences.

The *contrast* defines the difference between light and dark areas. It is the gain of the amplitude of luminance (Y) signal, see Figure 138. So gain>1 results in more contrast and gain<1 in less contrast.

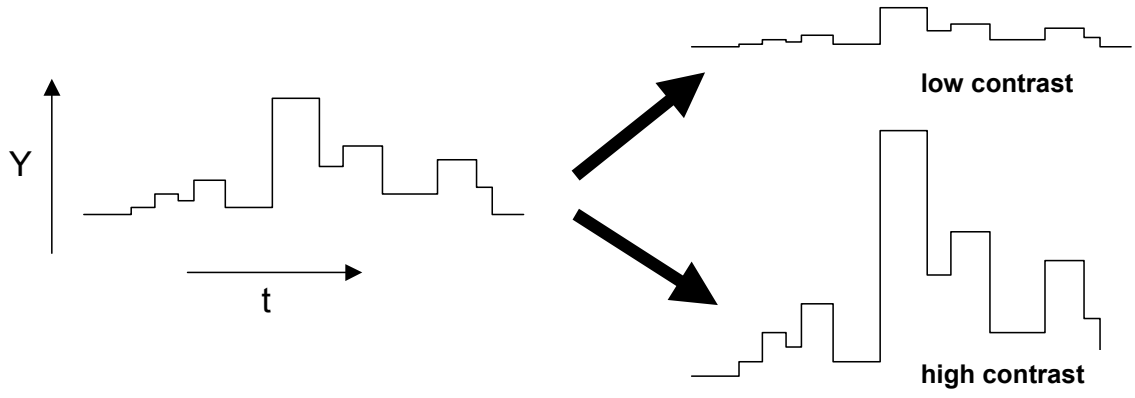


Figure 138 — Effect of Contrast control.

The *brightness* defines the overall (minimum) light level. It is controlled by adding an offset to the luminance (Y) signal, see Figure 139.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

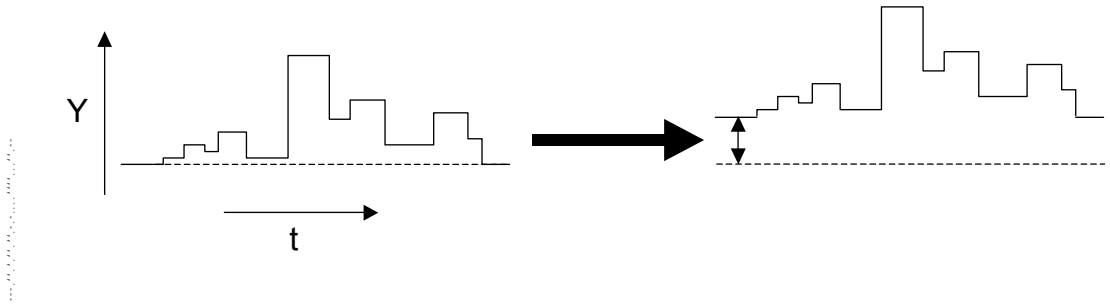


Figure 139 — Effect of Brightness control.

The *saturation* defines the amount of color added to the luminance (Y). It is the gain of the amplitude of the R-Y (≅V), G-Y, and B-Y(≅U) signals. So gain>1 results in more saturation and gain<1 in less.

The *hue* manages the shift of colors to compensate for the variable phase shifts in the modulated chrominance signal. This way shifts in colors are minimized.

The *white point* (also called tint or color temperature) defines the RGB value that is used to display the color white.

8.2.14.2 Types & Constants

8.2.14.2.1 Public Types & Constants

8.2.14.2.1.1 Error Codes

None.

8.2.14.2.1.2 `uhVfeat_BcsNotification_t`

Signature

```
typedef enum _uhVfeat_BcsNotification_t {
    uhVfeat_BcsSubscriptionChanged = 0x00000001,
    uhVfeat_BcsBrightnessChanged   = 0x00000002,
    uhVfeat_BcsContrastChanged     = 0x00000004,
    uhVfeat_BcsSaturationChanged   = 0x00000008
} uhVfeat_BcsNotification_t, *puhVfeat_BcsNotification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIVfeatBcsNtf`.

Values

Table 1158

Name	Description
`uhVfeat_BcsSubscriptionChanged`	Represents `OnSubscriptionChanged`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhVfeat_BcsBrightnessChanged	Represents OnBrightnessChanged.
uhVfeat_BcsContrastChanged	Represents OnContrastChanged.
uhVfeat_BcsSaturationChanged	Represents OnSaturationChanged.

8.2.14.2.1.3 uhVfeat_BcsNotifications_t

Signature
```
typedef Int32 uhVfeat_BcsNotifications_t, *puhVfeat_BcsNotifications_t;
```

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhVfeat_BcsNotification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.2.14.2.1.4 uhVfeat_BcsAllNotifications

Signature
```
const uhVfeat_BcsNotifications_t uhVfeat_BcsAllNotifications = 0x0000000F;
```

Qualifiers
None.

Description
Defines the set of all values of type uhVfeat_BcsNotification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.2.14.2.1.5 uhVfeat_HueNotification_t

Signature
```
typedef enum _uhVfeat_HueNotification_t {
    uhVfeat_HueSubscriptionChanged = 0x00000001,
    uhVfeat_HueChanged             = 0x00000002
} uhVfeat_HueNotification_t, *puhVfeat_HueNotification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhIVfeatHueNtf.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1159

Name	Description
uhVfeat_HueSubscriptionChanged	Represents OnSubscriptionChanged.
uhVfeat_HueChanged	Represents OnHueChanged.

8.2.14.2.1.6 uhVfeat_HueNotifications_t

Signature

```
typedef Int32 uhVfeat_HueNotifications_t, *puhVfeat_HueNotifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhVfeat_HueNotification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.2.14.2.1.7 uhVfeat_HueAllNotifications

Signature

```
const uhVfeat_HueNotifications_t uhVfeat_HueAllNotifications = 0x00000003;
```

Qualifiers

None.

Description

Defines the set of all values of type uhVfeat_HueNotification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.2.14.2.1.8 uhVfeat_WhitePointNotification_t

Signature

```
typedef enum _uhVfeat_WhitePointNotification_t {
    uhVfeat_WhitePointSubscriptionChanged = 0x00000001,
    uhVfeat_WhitePointChanged             = 0x00000002
} uhVfeat_WhitePointNotification_t, *puhVfeat_WhitePointNotification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface uhIVfeatWhitePointNtf.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1160

Name	Description
uhVfeat_WhitePointSubscriptionChanged	Represents OnSubscriptionChanged.
uhVfeat_WhitePointChanged	Represents OnWhitePointChanged.

8.2.14.2.1.9 uhVfeat_WhitePointNotifications_t

Signature

```
typedef Int32 uhVfeat_WhitePointNotifications_t,
    *puhVfeat_WhitePointNotifications_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhVfeat_WhitePointNotification_t. These values are used in the Subscribe and Unsubscribe functions to specify the event notifications to be enabled or disabled.

8.2.14.2.1.10 uhVfeat_WhitePointAllNotifications

Signature

```
const uhVfeat_WhitePointNotifications_t uhVfeat_WhitePointAllNotifications
    = 0x00000003;
```

Qualifiers

None.

Description

Defines the set of all values of type uhVfeat_WhitePointNotification_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.2.14.3 Logical Component

8.2.14.3.1 Interface-Role Model

Figure 140 depicts the interface-role model of the BVF logical component. It shows the control and notification interfaces and the roles involved with the BVF logical component.

-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

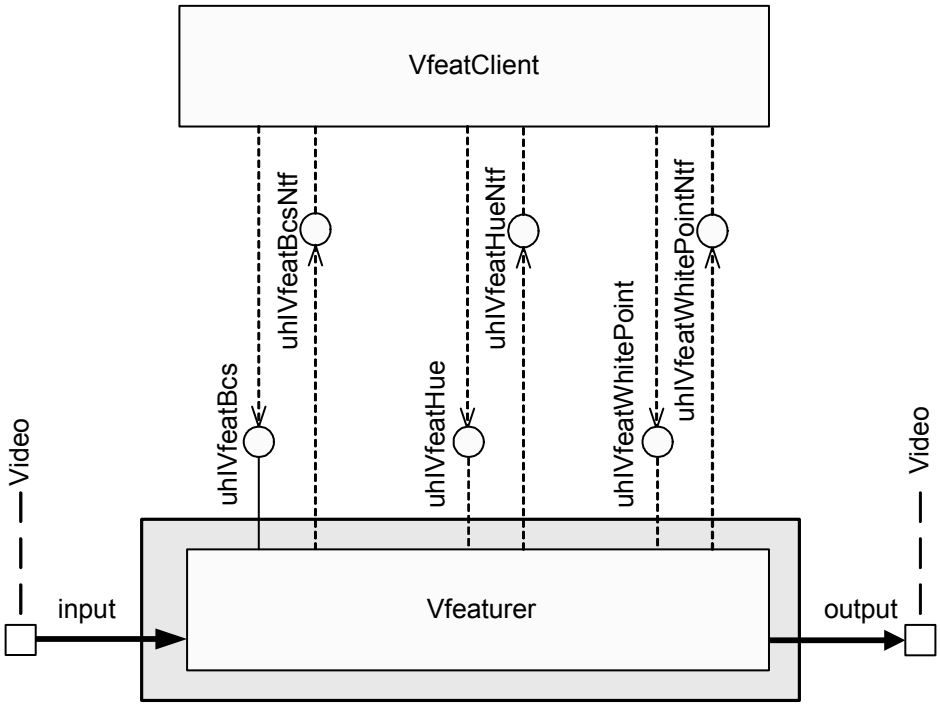


Figure 140 — Interface-Role Model

The grey box in Figure 140 denotes a BVF logical component containing a single `Vfeaturer` role. This role provides mandatory (solid line) and optional (dotted line) control interfaces to the clients.

Clients can optionally provide call-back interfaces allowing them to receive notifications of events in the `Vfeaturer`.

8.2.14.3.2 Diversity

8.2.14.3.2.1 Provided Interfaces

Table 1161

Role	Interface	Presence
`Vfeaturer`	`uhIVfeatBcs`	mandatory
`Vfeaturer`	`uhIVfeatHue`	optional
`Vfeaturer`	`uhIVfeatWhitePoint`	optional
`VfeatClient`	`uhIVfeatBcsNtf`	optional
`VfeatClient`	`uhIVfeatHueNtf`	optional
`VfeatClient`	`uhIVfeatWhitePointNtf`	optional

8.2.14.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of the logical component. For a description of the attributes see 8.2.14.4 Roles, and for how they are initialized see subclause 8.2.14.3.3.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1162

Role	Attribute
Vfeaturer	minBrightness
Vfeaturer	maxBrightness
Vfeaturer	brightness
Vfeaturer	minContrast
Vfeaturer	maxContrast
Vfeaturer	contrast
Vfeaturer	minSaturation
Vfeaturer	maxSaturation
Vfeaturer	saturation
Vfeaturer	minHue
Vfeaturer	maxHue
Vfeaturer	hue
Vfeaturer	minWhitePoint
Vfeaturer	maxWhitePoint
Vfeaturer	whitePoint

8.2.14.3.2.3 Constraints

See the Invariants of role Vfeaturer in subclause 8.2.14.4.1.

8.2.14.3.3 Instantiation

8.2.14.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1163

Type	Object	Multiplicity
Vfeaturer	vfeat	1

8.2.14.3.3.2 Initial State

The initial state of a logical component instance is completely determined by the values of the configurable items (see subclause 8.2.14.3.3.2).

8.2.14.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.14.4 Roles

8.2.14.4.1 Vfeaturer

Signature

```
role Vfeaturer {
    const UInt32          minBrightness;
    const UInt32          maxBrightness;
    UInt32                brightness;
    const UInt32          minContrast;
    const UInt32          maxContrast;
    UInt32                contrast;
    const UInt32          minSaturation;
    const UInt32          maxSaturation;
    UInt32                saturation;
    const UInt32          minHue;
    const UInt32          maxHue;
    UInt32                hue;
    const uhColor_t       minWhitePoint;
    const uhColor_t       maxWhitePoint;
    uhColor_t             whitePoint;
}
```

Qualifiers

— root

Description

A `Vfeaturer` controls the brightness, contrast, saturation, hue, and white point of the video input stream of a BVF logical component. Changes made via the interface are notified to the subscribed clients.

Independent Attributes

Table 1164

Name	Description
`minBrightness`	The minimum brightness.
`maxBrightness`	The maximum brightness.
`brightness`	The brightness.
`minContrast`	The minimum contrast.
`maxContrast`	The maximum contrast.
`contrast`	The contrast.
`minSaturation`	The minimum saturation.
`maxSaturation`	The maximum saturation.
`saturation`	The saturation.
`minHue`	The minimum hue.
`maxHue`	The maximum hue.
`hue`	The hue.
`minWhitePoint`	The maximum value for the white point in RGB format.
`maxWhitePoint`	The minimum value for the white point in RGB format.
whitePoint	The white point in RGB format.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes
None.

Invariants
— `minBrightness <=` brightness `<= maxBrightness`
— `minContrast <=` *contrast* `<= maxContrast`
— `minSaturation <=` saturation `<= maxSaturation`
— `minHue <=` *hue* `<= maxHue`
— `minWhitePoint <=25` whitePoint `<= maxWhitePoint`

Instantiation
A `Vfeaturer` role is always instantiated as part of a BVF logical component; see subclause 8.2.14.3.3.

Streaming Behavior
The `Vfeaturer` role changes the brightness, contrast, saturation, hue, and white point from the input stream as requested by the end user and sends the modified stream to its output.

Active Behavior
The `Vfeaturer role` has standard notification behavior (see `uhIVfeatBcsNtf`, `uhIVfeatHueNtf` and `uhIVfeatWhitePointNtf`).

8.2.14.4.2 VfeatClient

Signature
`role` `VfeatClient {}`

Qualifiers
— actor

Description
A `VfeatClient` role represents a user of the control interfaces offered by the `Vfeaturer` role. It can optionally provide the notification interfaces to receive notifications of events generated by the `Vfeaturer` role. No constraints apply to the behavior of the `VfeatClient` role other than those implied by Figure 140.

8.2.14.5 Interfaces

8.2.14.5.1 uhIVfeatBcs

Qualifiers
None.

Description
This interface of a `Vfeaturer` role provides control of the brightness, contrast, and saturation for a video input signal.

---

[25] The <= is an ordering on RGB colors, meaning x <= y implies x.R <= y.R and x.G <= y.G and x.B <= y.B.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
uuid(B968F010-E26B-49c4-A765-61E22C153C22)

Remarks
— In the following function specifications the name vfeat will be used to refer to the provider of this interface.

8.2.14.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIVfeatBcsNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVfeat_BcsNotifications_t notifs );
```

Qualifiers
— subscribe-function

8.2.14.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIVfeatBcsNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVfeat_BcsNotifications_t notifs );
```

Qualifiers
— unsubscribe-function

8.2.14.5.1.3 GetBrightnessRange

Signature
```
uhErrorCode_t GetBrightnessRange (
    [out] pUInt32 pMinBrightness,
    [out] pUInt32 pMaxBrightness );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and maximum brightness.

Parameters

Table 1165

Name	Description
pMinBrightness	Output parameter returning the minimum brightness.
pMaxBrightness	Output parameter returning the maximum brightness

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— **pMinBrightness* `==` `vfeat.minBrightness`
— *pMaxBrightness `==` `vfeat.maxBrightness`

8.2.14.5.1.4 GetBrightness

Signature
```
uhErrorCode_t GetBrightness ( [out] pUInt32 pBrightness );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the brightness.

Parameters

Table 1166

Name	Description
`pBrightness`	Output parameter returning the current brightness.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— **pBrightness* **==** `vfeat.brightness`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.14.5.1.5 SetBrightness

Signature
```
uhErrorCode_t SetBrightness ( [in] UInt32 brightness );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the brightness.

Parameters

Table 1167

Name	Description
`brightness`	The new brightness.

Return Values
Standard.

Precondition
— `vfeat.minBrightness` ≤ brightness ≤ `vfeat.maxBrightness`

Action
— Modify `vfeat.brightness`

Postcondition
— `vfeat.brightness` == *brightness*

8.2.14.5.1.6 GetContrastRange

Signature
```
uhErrorCode_t GetContrastRange (
    [out] pUInt32 pMinContrast,
    [out] pUInt32 pMaxContrast );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and maximum contrast.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1168

Name	Description
pMinContrast	Output parameter returning the minimum contrast.
pMaxContrast	Output parameter returning the maximum contrast.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— *pMinContrast `== vfeat.minContrast`
— ***pMaxContrast ==*** `vfeat.maxContrast`

8.2.14.5.1.7 GetContrast

Signature
```
uhErrorCode_t GetContrast ( [out] pUInt32 pContrast );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the contrast.

Parameters

Table 1169

Name	Description
pContrast	Output parameter returning the current contrast.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— *pContrast == vfeat.contrast

8.2.14.5.1.8 SetContrast

Signature
```
uhErrorCode_t SetContrast ( [in] UInt32 contrast );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the contrast.

Parameters

Table 1170

Name	Description
contrast	The new contrast.

Return Values
Standard.

Precondition
— vfeat.minContrast ≤ *contrast* ≤ vfeat.maxContrast

Action
Modify vfeat.contrast

Postcondition
— vfeat.contrast == contrast

8.2.14.5.1.9 GetSaturationRange

Signature
```
uhErrorCode_t GetSaturationRange (
    [out] pUInt32 pMinSaturation,
    [out] pUInt32 pMaxSaturation );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and maximum saturation.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1171

Name	Description
pMinSaturation	Output parameter returning the minimum saturation.
pMaxSaturation	Output parameter returning the maximum saturation.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— **pMinSaturation* == vfeat.minSaturation
— *pMaxSaturation == vfeat.maxSaturation

8.2.14.5.1.10 GetSaturation

Signature
```
uhErrorCode_t GetSaturation ( [out] pUInt32 pSaturation );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the saturation.

Parameters

Table 1172

Name	Description
pSaturation	Output parameter returning the current saturation.

Return Values
Standard.

Precondition
True

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— **pSaturation* == `vfeat.saturation`

8.2.14.5.1.11 SetSaturation

Signature
```
uhErrorCode_t SetSaturation ( [in] UInt32 saturation );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the saturation.

Parameters

Table 1173

Name	Description
`saturation`	The new saturation.

Return Values
Standard.

Precondition
— `vfeat.minSaturation` ≤ saturation ≤ `vfeat.maxSaturation`

Action
— Modify `vfeat.saturation`

Postcondition
— `vfeat.saturation` == *saturation*

8.2.14.5.2 uhIVfeatBcsNtf

Qualifiers
None.

Description
This interface can be provided by each instance of a `VfeatClient` role that wants to receive notifications of events that occur in the `vfeat`. The `Subscribe` function of `uhIVfeatBcs` can be used to subscribe to these events.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
uuid(E9FC44DA-F67E-423b-A09A-37B347308503)

8.2.14.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVfeat_BcsNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.14.5.2.2 OnBrightnessChanged

Signature
```
[async] Void OnBrightnessChanged (
    [in] UInt32 cookie,
    [in] UInt32 brightness );
```

Qualifiers
None.

Description
This function is called by a `Vfeaturer` whenever the brightness has changed, possibly via a `SetBrightness` call.

Parameters

Table 1174

Name	Description
`cookie`	The value that was passed by an instance of the `VfeatClient` role at subscription time.
`brightness`	The changed value of the brightness.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.14.5.2.3 OnContrastChanged

Signature

```
[async] Void OnContrastChanged (
    [in] UInt32 cookie,
    [in] UInt32 contrast );
```

Qualifiers
None.

Description
This function is called by a `Vfeaturer` whenever the contrast has changed, possibly via a `SetContrast` call.

Parameters

Table 1175

Name	Description
`cookie`	The value that was passed by an instance of the `VfeatClient` role at subscription time.
`contrast`	The changed value of the contrast.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.2.14.5.2.4 OnSaturationChanged

Signature

```
[async] Void OnSaturationChanged (
    [in] UInt32 cookie,
    [in] UInt32 saturation );
```

Qualifiers
None.

Description
This function is called by a `Vfeaturer` whenever the saturation has changed, possibly via a `SetSaturation` call.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1176

Name	Description
`cookie`	The value that was passed by an instance of the `VfeatClient` role at subscription time.
`saturation`	The changed value of the saturation.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.2.14.5.3 uhIVfeatHue

Qualifiers
None.

Description
This interface of `Vfeaturer` role provides control of the hue of a video signal.

Interface ID
`uuid( 3B481D7D-7C66-4ce7-986B-341001A7BEBE )`

Remarks
In the following function specifications the name `vfeat` will be used to refer to the provider of this interface.

8.2.14.5.3.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIVfeatHueNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVfeat_HueNotifications_t notifs );
```

Qualifiers
— subscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.14.5.3.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIVfeatHueNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVfeat_HueNotifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.2.14.5.3.3 GetHueRange

Signature

```
uhErrorCode_t GetHueRange (
    [out] pUInt32 pMinHue,
    [out] pUInt32 pMaxHue );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the minimum and maximum hue.

Parameters

Table 1177

Name	Description
`pMinHue`	Output parameter returning the minimum hue.
`pMaxHue`	Output parameter returning the maximum hue.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*`pMinHue `== vfeat.minHue`

— `*`*pMaxHue* `== vfeat.maxHue`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.14.5.3.4 GetHue

Signature
`uhErrorCode_t GetHue ( [out] pUInt32 pHue );`

Qualifiers
— synchronous

— thread-safe

Description
Returns the hue that was last set.

Parameters

Table 1178

Name	Description
`pHue`	Output parameter returning the hue.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*`pHue `== vfeat.hue`

8.2.14.5.3.5 SetHue

Signature
`uhErrorCode_t SetHue ( [in] UInt32 hue );`

Qualifiers
— synchronous

— single-threaded

Description
Sets the hue level.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1179

Name	Description
`hue`	The new hue.

Return Values
Standard.

Precondition
— `vfeat.minHue` ≤ *hue* ≤ `vfeat.maxHue`

Action
— `Modify vfeat.hue`

Postcondition
— `vfeat.hue ==` hue

8.2.14.5.4 uhIVfeatHueNtf

Qualifiers
None.

Description
This interface can be provided by an instance of the `VfeatClient` role that wants to receive notifications of events that occur in `vfeat`. The `Subscribe` function of the `uhIVfeatHue` interface can be used to subscribe to these events.

Interface ID
`uuid( 909EC152-78C6-4aab-B62B-8431EA8A2F8C )`

8.2.14.5.4.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVfeat_HueNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.14.5.4.2 OnHueChanged

Signature
```
[async] Void OnHueChanged (
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32 cookie,
    [in] UInt32 hue );
```

Qualifiers
None.

Description
This function is called by a `Vfeaturer` whenever the hue has changed, possibly via a `SetHue` call.

Parameters

Table 1180

Name	Description
`cookie`	The value that was passed by an instance of the `VfeatClient` role at subscription time.
`hue`	The changed value of the hue.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.2.14.5.5 uhlVfeatWhitePoint

Qualifiers
None.

Description
This interface of a `Vfeaturer` role provides control of the white point for a video input signal.

Interface ID
`uuid( B331A549-613E-4094-9E5A-E5891507E52E )`

Remarks
In the following function specifications the name `vfeat` will be used to refer to the provider of this interface.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.14.5.5.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (
    [in] uhIVfeatWhitePointNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVfeat_WhitePointNotifications_t notifs );
```

Qualifiers

— subscribe-function

8.2.14.5.5.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (
    [in] uhIVfeatWhitePointNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhVfeat_WhitePointNotifications_t notifs );
```

Qualifiers

— unsubscribe-function

8.2.14.5.5.3 GetWhitePointRange

Signature

```
uhErrorCode_t GetWhitePointRange (
    [out] puhColor_t pMinWhitePoint,
    [out] puhColor_t pMaxWhitePoint );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the minimum and maximum for the value of the white point in RGB format.

Parameters

Table 1181

Name	Description
`pMinWhitePoint`	Output parameter returning the minimum value of the white point in RGB format.
`pMaxWhitePoint`	Output parameter returning the maximum value of the white point in RGB format.

Return Values

Standard.

Precondition

`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— None.

Postcondition
— `*pMinWhitePoint == vfeat.minWhitePoint`
— `*pMaxWhitePoint == vfeat.maxWhitePoint`

8.2.14.5.5.4 GetWhitePoint

Signature
```
uhErrorCode_t GetWhitePoint ( [out] puhColor_t pWhitePoint );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the white point in RGB format.

Parameters

Table 1182

Name	Description
`pWhitePoint`	Output parameter returning the current white point in RGB format.

Return Values
Standard.

Precondition
`True`

Action
— None.

Postcondition
— `*pWhitePoint == vfeat.whitePoint`

8.2.14.5.5.5 SetWhitePoint

Signature
```
uhErrorCode_t SetWhitePoint ( [in] uhColor_t whitePoint );
```

Qualifiers
— synchronous

— single-threaded

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Sets the white point.

Parameters

Table 1183

Name	Description
`whitePoint`	The new white point in RGB format.

Return Values
Standard.

Precondition
— `vfeat.minWhitePoint <=26` whitePoint `<= vfeat.maxWhitePoint`

Action
— Modify `vfeat.whitePoint`

Postcondition
— `vfeat.whitePoint ==` *whitePoint*

8.2.14.5.6 uhIVfeatWhitePointNtf

Qualifiers
None.

Description
This interface can be provided by an instance of the `VfeatClient` role that wants to receive notifications of events that occur in `vfeat`. The `Subscribe` function of the `uhIVfeatWhitePoint` interface can be used to subscribe to these events.

Interface ID
`uuid( 17C7BFFD-5011-4e72-AD87-7B907114AF34 )`

8.2.14.5.6.1 OnSubscriptionChanged

Signature
`[async]` `Void OnSubscriptionChanged (`

[26] The <= is an ordering on RGB colors, meaning x <= y implies x.R <= y.R and x.G <= y.G and x.B <= y.B.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32 cookie,
    [in] uhVfeat_WhitePointNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.14.5.6.2 OnWhitePointChanged

Signature

```
[async] Void OnWhitePointChanged (
    [in] UInt32 cookie,
    [in] uhColor_t whitePoint );
```

Qualifiers
None.

Description
This function is called by a `Vfeaturer` whenever the white point has changed, possibly via a `SetWhitePoint` call.

Parameters

Table 1184

Name	Description
`cookie`	The value that was passed by an instance of the `VfeatClient` role at subscription time.
`whitePoint`	The changed value of the white point.

Return Values
None.

Precondition
`True`

Action
— Any call-back compliant action.

Postcondition
`True`

8.2.15 UhlVmix

8.2.15.1 Concepts

8.2.15.1.1 Introduction

The Video Mixer combines multiple video and graphics inputs into a single video output stream. All inputs have certain properties such as width and height, pixel format, etcetera.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the Video Mixer, the video and graphics inputs are associated with *layers*. The maximum number of layers is determined by the hardware and is fixed. The number of layers used depends on the use-case. The capabilities of a layer are different for video and graphics; hence there are video and graphics layers.

The order of the layers is important since it defines the final resulting output. Layers may partially overlap each other and their content may be fully or partially opaque or transparent. The input layers that are mixed can be scaled and positioned individually. The input layers do not necessarily have to be the same size and have the same pixel format[27]. Of course, certain hardware implementations may have more stringent rules than the ones described in the general model.

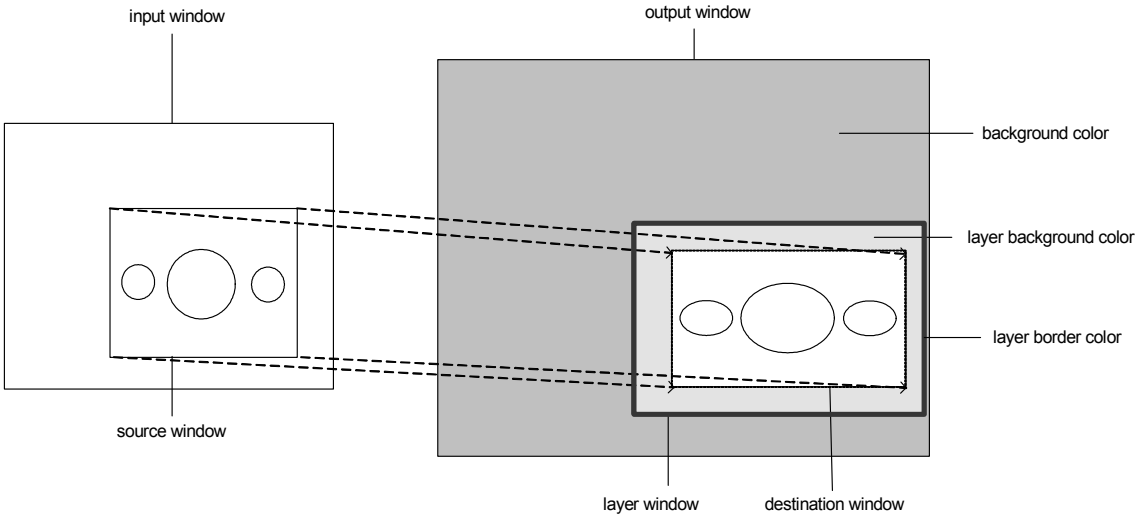


Figure 141 — The different windows of a layer.

Conceptually, it is convenient to discuss inputs, outputs, and layers in terms of windows. The following windows can be distinguished; see also Figure 141:

— An *input window*, which represents a complete input picture. It has the dimensions of the input that is received by the Video Mixer.

— A *source window*, which is used for clipping the input window. Pixels of the input window that are outside the source window will not be part of the video output stream. The source window does not need to be contained in the input window.

— An *output window*, which represents the complete video output stream. It has the dimensions of the output that is the result of the mixing process.

— A *layer window*, which defines the dimensions of a layer and the position of a layer with respect to the output window. A layer window can be positioned freely with respect to the output window, so part or all of the layer window may be outside the output window. The layer window's function is to provide a drawing plane for the destination window. Only pixels in both a layer window and the output window will be taken into account in the mixing process. Note that multiple layer windows exist in the Video Mixer, one for each layer. The layer window optionally supports borders and a background color (see Figure 142). If it doesn't support these, it can only be used for cropping the contents of the destination window.

— A *destination window*, which is used to scale the content of the source window to. A destination window is related to a layer window. It can be positioned freely with respect to the layer window, so part or all of the destination window may be outside the layer window (only pixels inside the layer window are taken into

[27] A pixel format can be for example YUV420.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

account in the mixing process). For all pixels outside a destination window but in its corresponding layer window, the layer background color or border color is inserted (if supported).

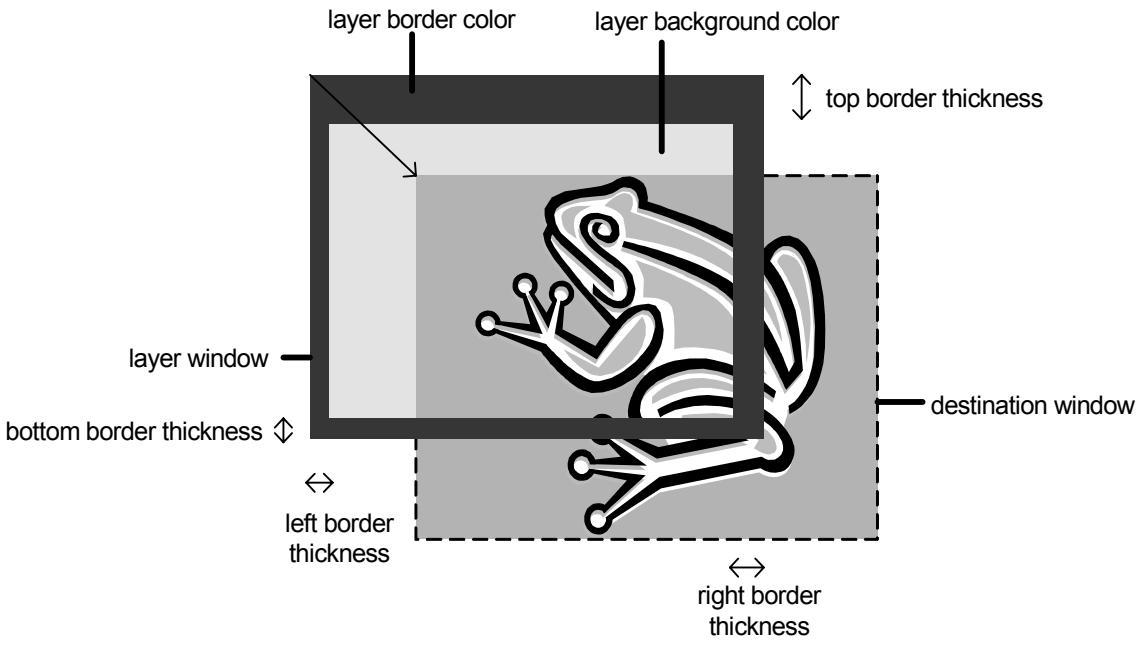


Figure 142 — Layer window borders.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

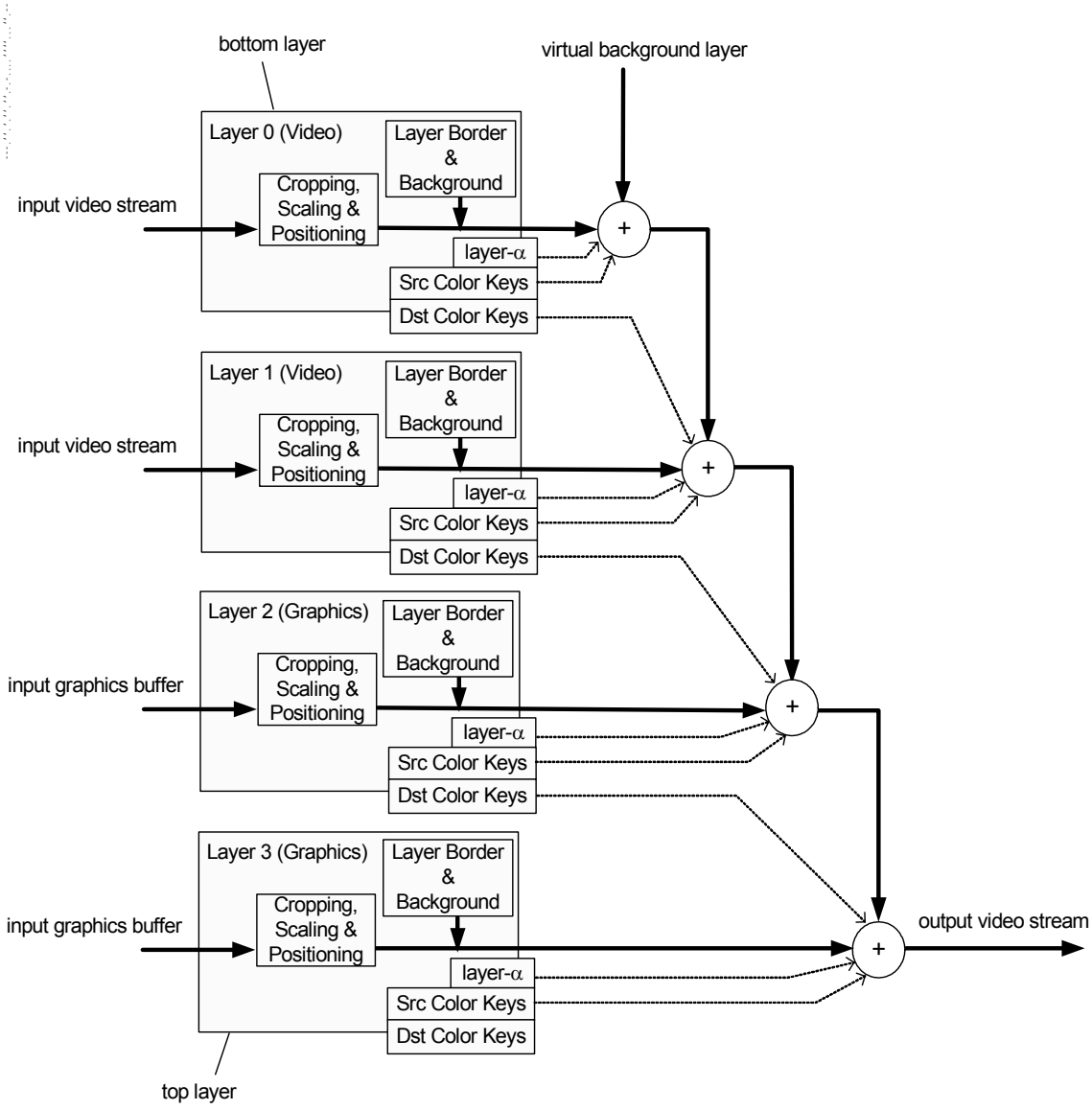


Figure 143 — The mixing process (in a mixer with two video layers and two graphics layers).

Figure 143 depicts the incremental mixing process. A special layer, called the *virtual background layer* defines pixels that all have the same color, and always has the same width and height as the video output stream. First, the virtual background layer is mixed with layer 0. The result is then mixed with layer 1, and so on. Prior to being mixed, each layer can crop, scale, and position its input, and give it a new pixel format, transparency level[28] (see subclause 8.2.15.1.6), and color key (see subclause 8.2.15.1.5). If supported, a layer can have its own background color and borders.

8.2.15.1.2 Blanking

Commonly blanking is making the screen black. For the Video Mixer, blanking can be done on the video output stream as well as on each layer. Blanking the video output stream means that the Video Mixer only displays a color specified by the client. Blanking a layer means that the entire layer window (within the borders, if present) shows a color specified by the client. Blanking can only be done on video layers. All layers (video and graphics) provide 'hide' functionality that takes the layer out of the mixing process.

[28] This level is for the entire layer, transparency of individual pixels is realized via the pixel format.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.1.3 Smooth zooming

The mixer supports smooth window movements such as zooming. For each window associated with a layer the client can specify a target window. After specifying the targets, the client can let the mixer move all windows smoothly to their respective targets in a specified time. The windows move in such a way that all corners arrive at their destination at the same time (so the corners move at varying speeds). See also Figure 144.

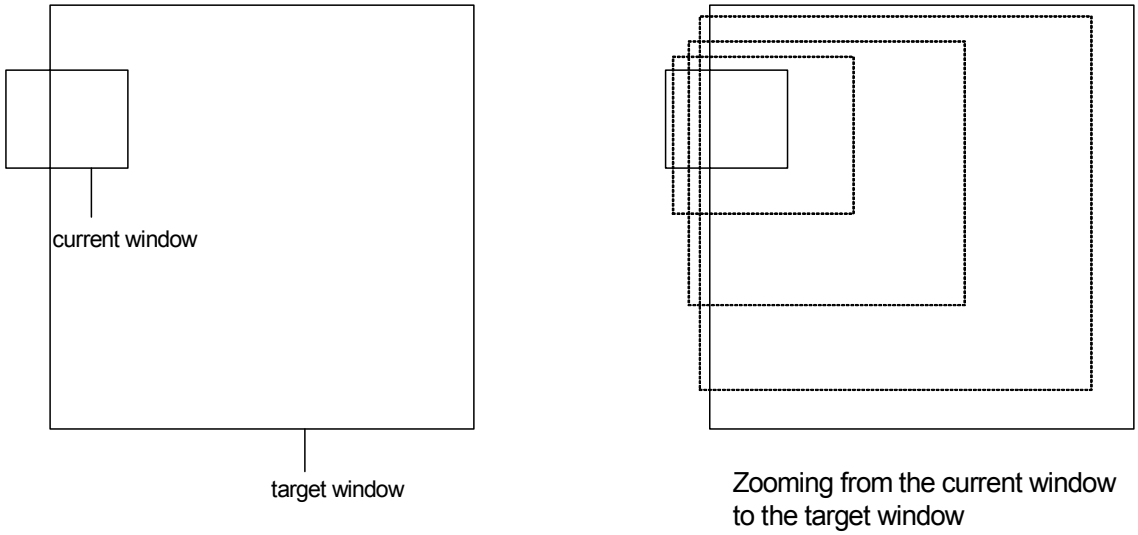


Figure 144 — Smooth zooming / moving of windows.

8.2.15.1.4 Panoramic scaling

Panoramic scaling is a form of scaling used to display images with a 4:3 or 14:9 aspect ratio on a 16:9 television set. Figure 145 below shows the purpose of panoramic scaling. The side of the picture is stretched more than the middle. As you can see, the circle that we get with panoramic scaling is still deformed but less than the normal stretched circle (linear scaling). So in the middle of the picture there is less distortion of the aspect ratio. Since most of the time the actual interesting part of the scene 'happens' in the middle, this panoramic distortion is generally perceived as less disturbing than the distortion due to linear scaling.

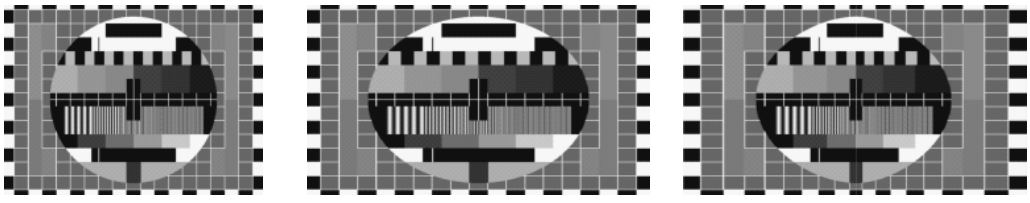


Figure 145 — A 4:3 image scaled linearly (stretched) and in panoramic mode to 16:9.

A platform instance does not necessarily support panoramic scaling, but when it does, the amount of distortion can be programmed by the client. The variable stretch and compression factors applied in panoramic scaling can be modeled by a parabola (see Figure 146). In this example the scaling factor in the middle of the screen is approximately 1 (100%), and towards both sides of the screen stretching increases up to a factor 2 (200%). The client can tune this parabola to some extent (displaying a 14:9 image nicely on a 16:9 screen requires a different parabola than the one used for 4:3 images). One can also do the "inverse" of panoramic scaling (amaronap); see Figure 146: little scaling in the middle, and a lot of compression on the side. This variant can be used to display a 16:9 image on a 4:3 screen.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

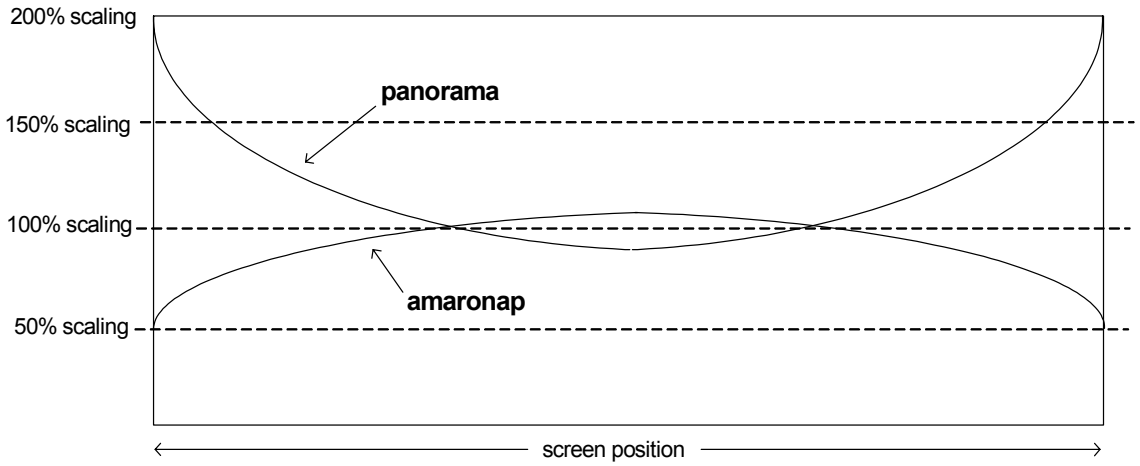


Figure 146 — The parabola defining the variable stretch/compression factors used in panoramic scaling.

8.2.15.1.5 Color-keying

In Color Keying pixels with a specific color are replaced by pixels of another picture or a color from a color generator. The most common form of color keying is called Blue Screening (as is used in making movies and weather reports). It is a simple operation, which replaces a pixel in the foreground layer with a pixel from the background layer when the color of the foreground layer pixel is within a specified range. For example, an actor is filmed in front of a blue background and later a different background is mixed in by replacing the blue pixels from the recording of the actor.

In the video domain, blue screening can be used for showing a video window in a graphics window. When each window is in a separate layer, the hardware can do the mixing and the software only has to do the positioning. This is more efficient than doing the mixing in software.

Blue Screening can also be achieved using Per Pixel Alpha Blending (see subclause 8.2.15.1.6). Note that, this may result in a larger memory requirement when an extra byte per pixel is required for the alpha value.

The video mixer supports multiple color keys per layer. The color keys used in the mixer are in fact color key ranges (identified by a lower and an upper color value; RGB on a graphics layer and YUV on a video layer). If a non-border pixel of a layer falls in one of the ranges defined by that layer, it will generate a color key hit. There are two types of color key and they differ in how color key hits are processed:

— *Source Color Keying* : this is the form of color keying called Blue Screening above. Pixels generating a hit will be completely transparent when mixed.

— *Destination Color Keying* : with this form of color keying, a mask can be created for the next layer. Only those pixels of the next layer are mixed that coincide with pixels generating a hit in the current layer (see Figure 147 for an example).

It is implementation defined what happens if a pixel generates both a source and a destination color key hit on the same layer, or if a pixel on some layer generates a destination color key hit, while the corresponding pixel on the next layer generates a source color key hit.

Pixels which generate a *destination color key hit* can be visible (directly) in the end result of mixing because the layer for which the mask is intended might not cover these pixels completely. Pixels which generate a *source color key hit* are never directly visible in the end result but their color could affect the colors of neighbouring pixels as a result of filtering. Therefore, after a color key hit is generated, the mixer will replace such a pixel with some non-offensive color (e.g. gray).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

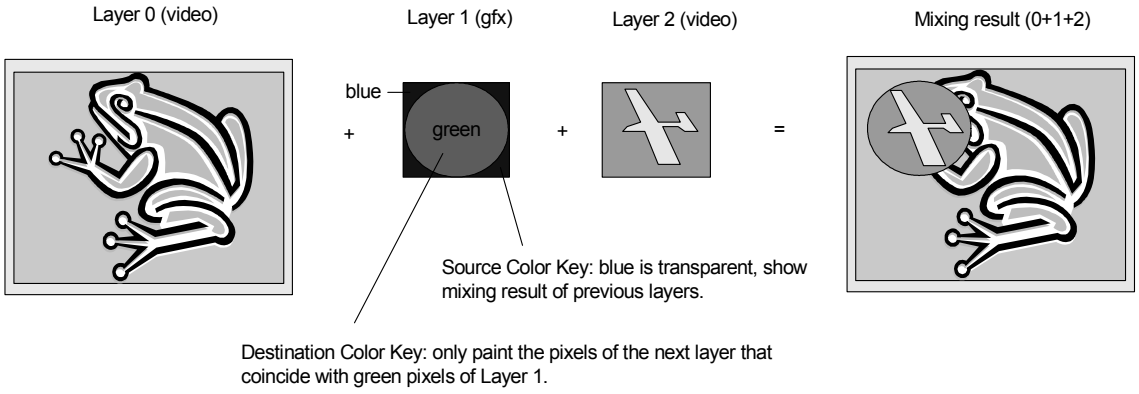


Figure 147 — Creating a round PIP with Source and Destination Color Keying.

8.2.15.1.6 Alpha blending

Alpha blending is combining overlapping, partially or completely transparent, pixels into a new pixel value. Alpha blending can be performed *per pixel* or *per layer*. Blending *per pixel* means that each pixel in a layer has its own transparency value (0 <= pixelα <=1). Blending *per layer* applies the same transparency value (0<= globalα <=1) to all pixels of that layer except the pixels of the borders (note that though borders can not be transparent, alpha blending does apply to them in the reverse situation, where a (partially) transparent layer is on top of a layer with borders). In both cases, all pixel *components* (R,G, and B, or Y, U, and V) are multiplied with the same alpha value.

The result of *per layer* alpha blending is computed as follows (where x is the layer number; x is on top of x-1):

$$pixel_{result\ x} = (global\alpha_{layer\ x} * pixel_{layer\ x}) +$$

$$((1 - global\alpha_{layer\ x}) * pixel_{result\ x-1})$$

The result of *per pixel* alpha blending is computed as follows:

$$pixel_{result\ x} = (pixel\alpha_{layer\ x} * pixel_{layer\ x}) +$$

$$((1 - pixel\alpha_{layer\ x}) * pixel_{result\ x-1})$$

When both *per layer* and *per pixel* alpha blending are enabled, the result will be as follows:

$$\alpha layer\ x = pixel\alpha_{layer\ x} * global\alpha_{layer\ x}$$

$$pixel_{result\ x} = (\alpha_{layer\ x} * pixel_{layer\ x}) +$$

$$((1 - \alpha_{layer\ x}) * pixel_{result\ x-1})$$

When *per layer* or *per pixel* alpha blending is used in conjunction with color-key replacement, alpha blending will only be done for non-color key hits. The pixels that replace a color key hit are displayed opaque. Only after color key hits are resolved, meaning that a previous mixing phase already replaced them by pixels without transparency value, they will be taken into account for alpha blending during a next mixing phase.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.2 Types & Constants

8.2.15.2.1 Public Types & Constants

8.2.15.2.1.1 Error Codes

Signature

```
const uhErrorCode_t UH_ERR_VMIX_FADING           = 0x00000801;
const uhErrorCode_t UH_ERR_VMIX_SETTING_WINDOWS  = 0x00000802;
```

Qualifiers

— error-codes

Description

Defines the error codes returned by the functions in the interfaces of a Video Mixer.

Constants

Table 1185

Name	Description
UH_ERR_VMIX_FADING	The layer is currently fading (warning returned when one queries the layer's blending factor during a fade).
UH_ERR_VMIX_SETTING_WINDOWS	The layer is currently moving its windows (warning returned when one queries the layer's windows or scale mode).

8.2.15.2.1.2 uhVmix_ValStat_t

Signature

```
typedef enum _uhVmix_ValStat_t {
    uhVmix_ValStatInvalid = -2,
    uhVmix_ValStatUnknown = -1,
    uhVmix_ValStatKnown   =  1
} uhVmix_ValStat_t, *puhVmix_ValStat_t;
```

Qualifiers

None.

Description

This type represents the validity status of the related property/properties.

Values

Table 1186

Name	Description
uhVmix_ValStatInvalid	The property/properties are temporarily not available.
uhVmix_ValStatUnknown	The property/properties are available, yet they could not be determined.
uhVmix_ValStatKnown	The property/properties are available and determined.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.2.1.3 uhVmix_LayerType_t

Signature

```
typedef enum _uhVmix_LayerType_t
{
    uhVmix_VideoLayer = 0x00000001,
    uhVmix_GfxLayer   = 0x00000002
} uhVmix_LayerType_t, *puhVmix_LayerType_t;
```

Qualifiers
None.

Description
Defines the type of a layer. This type is used when returning the platform layer configuration.

Values

Table 1187

Name	Description
uhVmix_VideoLayer	Layer is a video layer.
uhVmix_GfxLayer	Layer is a graphics layer.

8.2.15.2.1.4 uhVmix_Notification_t

Signature

```
typedef enum _uhVmix_Notification_t
{
    uhVmix_SubscriptionChanged    = 0x00000001,
    uhVmix_NewProperties          = 0x00000002
} uhVmix_Notification_t, *puhVmix_Notification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions in `uhIVmixNtf`. Can be used to individually turn on and off notifications.

Values

Table 1188

Name	Description
uhVmix_LayerSubscriptionChanged	Represents `OnSubscriptionChanged`.
uhVmix_NewProperties	Represents `OnNewProperties`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.2.1.5 uhVmix_Notifications_t

Signature
```
typedef Int32 uhVmix_Notifications_t, *puhVmix_Notifications_t;
```

Qualifiers
— enum-set

Description
Defines the set-type for `uhVmix_Notification_t`.

8.2.15.2.1.6 uhVmix_AllNotifications

Signature
```
const uhVmix_Notifications_t uhVmix_AllNotifications     = 0x00000003;
```

Qualifiers
None.

Description
The set of all notifications of type `uhVmix_Notification_t`. This constant can be used in calls of *Subscribe* and *Unsubscribe* functions to enable or disable all event notifications, respectively.

8.2.15.2.1.7 uhVmix_LayerNotification_t

Signature
```
typedef enum _uhVmix_LayerNotification_t
{
    uhVmix_LayerSubscriptionChanged      = 0x00000001,
    uhVmix_LayerFadeInterrupted          = 0x00000002,
    uhVmix_LayerFadeCompleted            = 0x00000004,
    uhVmix_LayerNewSettingsInterrupted   = 0x00000008,
    uhVmix_LayerNewSettingsCompleted     = 0x00000010
} uhVmix_LayerNotification_t, *puhVmix_LayerNotification_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions in `uhIVmixLayerNtf`. Can be used to individually turn on and off notifications.

Values

Table 1189

Name	Description
`uhVmix`_LayerSubscriptionChanged	Represents `OnSubscriptionChanged`.
`uhVmix`_LayerFadeInterrupted	Represents `OnFadeInterrupted`.
`uhVmix`_LayerFadeCompleted	Represents `OnFadeCompleted`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhVmix_LayerNewSettingsInterrupted	Represents OnNewSettingsInterrupted.
uhVmix_LayerNewSettingsCompleted	Represents OnNewSettingsCompleted.

8.2.15.2.1.8 uhVmix_LayerNotifications_t

Signature

```
typedef Int32 uhVmix_LayerNotifications_t, *puhVmix_LayerNotifications_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhVmix_LayerNotification_t.

8.2.15.2.1.9 uhVmix_LayerAllNotifications

Signature

```
const uhVmix_LayerNotifications_t uhVmix_LayerAllNotifications = 0x0000001F;
```

Qualifiers

None.

Description

The set of all notifications of type uhVmix_LayerNotification_t. This constant can be used in calls of *Subscribe* and *Unsubscribe* functions to enable or disable all event notifications, respectively.

8.2.15.2.1.10 uhVmix_VidLayerNotification_t

Signature

```
typedef enum _uhVmix_VidLayerNotification_t
{
    uhVmix_VidLayerSubscriptionChanged = 0x00000001,
    uhVmix_VidLayerAutoUnfrozen        = 0x00000002,
    uhVmix_VidLayerFrameCaptured       = 0x00000004
} uhVmix_VidLayerNotification_t, *puhVmix_VidLayerNotification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions in uhIVmix_VidLayerNtf. Can be used to individually turn on and off notifications.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1190

Name	Description
uhVmix_VidLayerSubscriptionChanged	Represents OnSubscriptionChanged.
uhVmix_VidLayerAutoUnfrozen	Represents OnAutoUnfrozen.
uhVmix_VidLayerFrameCaptured	Represents OnFrameCaptured.

8.2.15.2.1.11 uhVmix_VidLayerNotifications_t

Signature

```
typedef Int32 uhVmix_VidLayerNotifications_t, *puhVmix_VidLayerNotifications_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhVmix_VidLayerNotification_t.

8.2.15.2.1.12 uhVmix_VidLayerAllNotifications

Signature

```
const uhVmix_VidLayerNotifications_t
    uhVmix_VidLayerAllNotifications = 0x00000007;
```

Qualifiers

None.

Description

The set of all notifications of type uhVmix_VidLayerNotification_t. This constant can be used in calls of *Subscribe* and *Unsubscribe* functions to enable or disable all event notifications, respectively.

8.2.15.2.1.13 uhVmix_GfxLayerNotification_t

Signature

```
typedef enum _uhVmix_GfxLayerNotification_t
{
    uhVmix_GfxLayerSubscriptionChanged = 0x00000001,
    uhVmix_GfxLayerBufferFree          = 0x00000002
} uhVmix_GfxLayerNotification_t, *puhVmix_GfxLayerNotification_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions in uhIVmixGfxLayerNtf. Can be used to individually turn on and off notifications.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1191

Name	Description
uhVmix_GfxLayerSubscriptionChanged	Represents OnSubscriptionChanged.
uhVmix_GfxLayerBufferFree	Represents OnBufferFree.

8.2.15.2.1.14 uhVmix_GfxLayerNotifications_t

Signature

```
typedef Int32 uhVmix_GfxLayerNotifications_t, *puhVmix_GfxLayerNotifications_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhVmix_GfxLayerNotification_t.

8.2.15.2.1.15 uhVmix_GfxLayerAllNotifications

Signature

```
const uhVmix_GfxLayerNotifications_t
    uhVmix_GfxLayerAllNotifications  = 0x00000003;
```

Qualifiers

None.

Description

The set of all notifications of type uhVmix_GfxLayerNotification_t. This constant can be used in calls of *Subscribe* and *Unsubscribe* functions to enable or disable all event notifications, respectively.

8.2.15.2.1.16 uhVmix_ScaleMode_t

Signature

```
typedef enum _uhVmix_ScaleMode_t
{
    uhVmix_NoScaling  = 0x00000001,
    uhVmix_Linear     = 0x00000002,
    uhVmix_Panorama   = 0x00000004,
    uhVmix_Horizontal = 0x00000008
} uhVmix_ScaleMode_t, *puhVmix_ScaleMode_t;
```

Qualifiers

— enum-element

Description

Defines the possible scale modes.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1192

Name	Description
uhVmix_NoScaling	No scaling.
uhVmix_Linear	Linear horizontal and vertical scaling.
uhVmix_Panorama	Non-linear horizontal and linear vertical scaling meant for displaying 4:3 or 14:9 images on a 16:9 display.
uhVmix_Horizontal	Linear horizontal scaling and no vertical scaling.

8.2.15.2.1.17 uhVmix_ScaleModeSet_t

Signature

```
typedef Int32 uhVmix_ScaleModeSet_t, *puhVmix_ScaleModeSet_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhVmix_ScaleMode_t.

8.2.15.2.1.18 uhVmix_FieldRate_t

Signature

```
typedef enum _uhVmix_FieldRate_t
{
    uhVmix_FieldRateInvalid = -2,
    uhVmix_FieldRateUnknown = -1,
    uhVmix_50Hz             =  0,
    uhVmix_60Hz             =  1,
    uhVmix_75Hz             =  2,
    uhVmix_100Hz            =  3
} uhVmix_FieldRate_t, *puhVmix_FieldRate_t;
```

Qualifiers

— enum

Description

Defines the field/frame rate of a (interlaced/progressive) video signal.

Values

Table 1193

Name	Description
uhVmix_FieldRateInvalid	The field rate property is temporarily not available.
uhVmix_FieldRateUnknown	The field rate property is available, yet it could not be determined.
uhVmix_50Hz	50 Hz.
uhVmix_60Hz	60 Hz.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhVmix_75Hz	75 Hz.
uhVmix_100Hz	100 Hz.

8.2.15.2.1.19 uhVmix_ScanType_t

Signature

```
typedef enum _uhVmix_ScanType_t
{
    uhVmix_ScanTypeInvalid = -2,
    uhVmix_ScanTypeUnknown = -1,
    uhVmix_Interlaced      =  0,
    uhVmix_Progressive     =  1
} uhVmix_ScanType_t, *puhVmix_ScanType_t;
```

Qualifiers

— enum

Description

Defines the possible scan types.

Values

Table 1194

Name	Description
uhVmix_ScanTypeInvalid	The scan type property is temporarily not available.
uhVmix_ScanTypeUnknown	The scan type property is available, yet it could not be determined.
uhVmix_Interlaced	Interlaced scanning.
uhVmix_Progressive	Progressive scanning.

8.2.15.2.1.20 uhVmix_ColorKeyType_t

Signature

```
typedef enum _uhVmix_ColorKeyType_t
{
    uhVmix_SrcColorKey     = 0x00000001,
    uhVmix_DstColorKey     = 0x00000002
} uhVmix_ColorKeyType_t, *puhVmix_ColorKeyType_t;
```

Qualifiers

— enum-element

Description

Defines the different types of color keying. See subclause 8.2.15.1.5 for the definitions of source and destination color keying.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Table 1195

Name	Description
uhVmix_SrcColorKey	Source color key.
uhVmix_DstColorKey	Destination color key.

8.2.15.2.1.21 uhVmix_ColorKeyTypeSet_t

Signature

```
typedef Int32 uhVmix_ColorKeyTypeSet_t, *puhVmix_ColorKeyTypeSet_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for uhVmix_ColorKeyType_t.

8.2.15.2.1.22 uhVmix_NewVidLayerPropChanged_t

Signature

```
typedef enum _uhVmix_NewVidLayerPropChanged_t
{
    uhVmix_ResolutionChanged       = 0x00000001,
    uhVmix_AspectRatioChanged      = 0x00000002,
    uhVmix_ActiveImageAreaChanged  = 0x00000004
} uhVmix_NewVidLayerPropChanged_t, *puhVmix_NewVidLayerPropChanged_t;
```

Qualifiers

— enum-element

Description

Values of this type are indications that a certain property of the input of a video layer has changed.

Values

Table 1196

Name	Description
uhVmix_ResolutionChanged	Indication that the input resolution of the video layer has changed.
uhVmix_AspectRatioChanged	Indication that the aspect ratio of the input of the video layer has changed.
uhVmix_ActiveImageAreaChanged	Indication that the active image area (the actual video, excluding black bars) of the input of the video layer has changed.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.2.1.23 uhVmix_NewVidLayerPropChangedSet_t

Signature

```
typedef                    Int32                    uhVmix_NewVidLayerPropChangedSet_t,
*puhVmix_NewVidLayerPropChangedSet_t;
```

Qualifiers

— enum-set

Description

Defines the set-type for `uhVmix_NewVidLayerPropChanged_t`.

8.2.15.2.1.24 uhVmix_NewVidLayerProp_t

Signature

```
typedef struct _uhVmix_NewVidLayerProp_t {
    uhVmix_NewVidLayerPropChangedSet_t  changes;
    uhVmix_ValStat_t                    status;
    UInt32                              horResolution;
    UInt32                              vertResolution;
    UInt16                              aspectRatioNumerator;
    UInt16                              aspectRatioDenominator;
    uhRect_t                            activeImageArea;
} uhVmix_NewVidLayerProp_t, *puhVmix_NewVidLayerProp_t;
```

Qualifiers

None.

Description

Values of this type contain the properties of the input of a video layer. The type is used to notify video property changes to clients.

Fields

Table 1197

Name	Description
`changes`	Indicates which of the properties of the video layer input have changed.
`status`	Validity state of the video layer properties (horizontal/vertical resolution, aspect ratio numerator/denominator, active image area).
`horResolution`	The horizontal resolution of the video layer input.
`vertResolution`	The vertical resolution of the video layer input.
`aspectRatioNumerator`	Aspect ratio of the video layer input is aspectRatioNumerator:aspectRatioDenominator. If there is no aspect ratio information in the signal, `aspectRatioNumerator == 0`.
`aspectRatioDenominator`	Aspect ratio of the video layer input is aspect`RatioNumerator:aspectRatioDenominator`. If there is no aspect ratio information in the signal, `aspectRatioDenominator == 1`.
`activeImageArea`	Active image area of the video layer input. If no information on the active image area is present in the signal (WSS data), this value is (0,0,0,0).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.2.1.25 uhVmix_NewGfxLayerProp_t

Signature

```
typedef struct _uhVmix_NewGfxLayerProp_t {
    UInt32 horResolution;
    UInt32 vertResolution;
} uhVmix_NewGfxLayerProp_t, *puhVmix_NewGfxLayerProp_t;
```

Qualifiers

None.

Description

Values of this type contain the properties of the input of a graphics layer. The type is used to notify input graphics property changes to clients.

Fields

Table 1198

Name	Description
horResolution	The horizontal resolution of the graphics layer input.
vertResolution	The vertical resolution of the graphics layer input.

8.2.15.2.2 Model Types & Constants

8.2.15.2.2.1 Borders

Signature

```
typedef struct {
    UInt32    top;
    UInt32    bottom;
    UInt32    left;
    UInt32    right;
    Bool      hidden;
    uhColor_t color;
} Borders;
```

Qualifiers

None.

Description

Values of this type represent the aspects of the borders of a layer.

Fields

Table 1199

Name	Description
top	Thickness (in lines) of the top border.
bottom	Thickness (in lines) of the bottom border.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
left	Thickness (in pixels) of the left border.
right	Thickness (in pixels) of the right border.
hidden	The borders are visible (False) or hidden (True).
color	The color of the borders.

8.2.15.2.2.2 LayerSettings

Signature

```
typedef struct {
    uhRect_t             src;
    uhRect_t             dst;
    uhRect_t             layer;
    uhVmix_ScaleMode_t   scaleMode;
    UInt32               panoramaMidScaleRatio;
} LayerSettings;
```

Qualifiers
None.

Description
Values of this type represent a number of settings of a layer that can be effectuated simultaneously by calling uhIVmixLayer_ActivateNewWindowSettings.

Fields

Table 1200

Name	Description
src	The source window.
dst	The destination window.
layer	The layer window.
scaleMode	The scale mode.
panoramaMidScaleRatio	A parameter for tuning the panoramic scaling parabola: the scale ratio in the middle of the screen (in percentages; panoramic: < 100 %, linear: 100%, amaronap: > 100 %).

8.2.15.2.2.3 GfxBuffer

Signature

```
typedef struct {
    pUInt8          pContents;
    UInt32          width;
    UInt32          height;
} GfxBuffer;
```

Qualifiers
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent a graphics buffer.

Fields

Table 1201

Name	Description
pContents	A pointer to the buffer (contents).
width	Width of the buffer in pixels.
height	Height of the buffer in lines.

8.2.15.2.2.4 BlendingRange

Signature
```
typedef struct {
    UInt32        min;
    UInt32        max;
} BlendingRange;
```

Qualifiers
None.

Description
Values of this type represent a blending range. An integer b in such a range represents the alpha blending value α = (b - min) / (max - min).

Fields

Table 1202

Name	Description
min	The minimum blending factor (corresponding to fully transparent).
max	The maximum blending factor (corresponding to fully opaque).

8.2.15.2.2.5 OutputProperties

Signature
```
typedef struct {
    uhVmix_ValStat_t     status;
    UInt32               width;
    UInt32               height;
    uhVmix_FieldRate_t   fieldRate;
    uhVmix_ScanType_t    scanType;
} OutputProperties;
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Values of this type represent the (run-time) configurable properties of the output video.

Fields

Table 1203

Name	Description
`status`	Validity state of the video width and height properties.
`width`	The width (pixels per line) of the output video.
`height`	The height (lines) of the output video.
`fieldRate`	The field/frame rate of the output video.
`scanType`	The scan type (interlaced, progressive, etc.) of the output video.

8.2.15.2.2.6 InputProperties

Signature

```
typedef struct {
    UInt32              width;
    UInt32              height;
    UInt16              aspectRatioNumerator;
    UInt16              aspectRatioDenominator;
    uhRect_t            activeImageArea;
} InputProperties;
```

Qualifiers
None.

Description
Values of this type represent the input properties of a video layer, i.e. the resolution, the pixel aspect ratio, and the active image area, which is the part of the picture that actually contains video (information contained in WSS data).

Fields

Table 1204

Name	Description
`width`	The width (pixels per line) of the input window.
`height`	The height (lines) of the input window.
`aspectRatioNumerator`	The aspect ratio of the input is aspect`RatioNumerator:aspectRatioDenominator`. If there is no information on the aspect ratio available in the input, `aspectRatioNumerator == 0`.
`aspectRatioDenominator`	The aspect ratio of the input is `aspectRatioNumerator:aspectRatioDenominator`. If there is no information on the aspect ratio available in the input, `aspectRatioDenominator == 1`.
`activeImageArea`	The active image area of the input. If the input does not contain information on the active image area, `activeImageArea == (0,0,0,0)`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.2.2.7 WindowSize

Signature

```
typedef struct {
    UInt32              width;
    UInt32              height;
} WindowSize;
```

Qualifiers
None.

Description
Values of this type represent the size (width, height) of a window.

Fields

Table 1205

Name	Description
`width`	The width (pixels per line) of the window.
`height`	The height (lines) of the window.

8.2.15.2.2.8 ScaleRange

Signature

```
typedef struct {
    UInt32              minWidthPercentage;
    UInt32              minHeightPercentage;
    UInt32              maxWidthPercentage;
    UInt32              maxHeightPercentage;
} ScaleRange;
```

Qualifiers
None.

Description
Values of this type represent the maximum and minimum, horizontal and vertical scaling factors in percentages. For instance, if minHeightPercentage == 50 and maxHeightPercentage == 200, then the height of the destination window (*dst height*) and the height of the source window (*src height*) must satisfy the following constraint: ½ * *src height* <= *dst height* <= 2 * *src height*.

Fields

Table 1206

Name	Description
`minWidthPercentage`	The minimum horizontal scale factor (in percentages).
`minHeighthPercentage`	The minimum vertical scale factor (in percentages).
`maxWidthPercentage`	The maximum horizontal scale factor (in percentages).
`maxHeightPercentage`	The maximum vertical scale factor (in percentages).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.2.2.9 PanoramaScaleRange

Signature

```
typedef struct {
    UInt32        min;
    UInt32        max;
 } PanoramaScaleRange;
```

Qualifiers
None.

Description
Values of this type represent a range of scaling factors (in percentages) that may be used for setting the mid scale ratio that defines the panoramic scaling parabola (together with the source and destination windows).

See Figure 156 for details.

Fields

Table 1207

Name	Description
min	The minimum scale factor (in percentages) that may be applied at the middle of the source window.
max	The maximum scale factor (in percentages) that may be applied at the middle of the source window.

8.2.15.2.2.10 PanoramaMidScaleRatioRange

Signature

```
typedef struct {
    UInt32        min;
    UInt32        max;
 } PanoramaMidScaleRatioRange;
```

Qualifiers
None.

Description
Values of this type represent a range of scaling factors (in percentages) that may be used for setting the mid scale ratio that defines the panoramic scaling parabola (together with the source and destination windows).

See Figure 155 and Figure 156 for details.

Fields

Table 1208

Name	Description
min	The minimum scale ratio (in percentages) that may be applied at the middle of the source window.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
max	The maximum scale ratio (in percentages) that may be applied at the middle of the source window.

8.2.15.2.2.11 ColorKey

Signature

```
typedef struct {
   uhVmix_ColorKeyType_t keyType;
   uhColor_t             lowerColor;
   uhColor_t             upperColor;
   Bool                  enabled;
} ColorKey;
```

Qualifiers
None.

Description
Values of this type represent a color key (range). The range limits lowerColor and upperColor are included in the range. These color values are interpreted as RGB on graphics layers and YUV on video layers. See subclause 8.2.15.1.5 for an explanation of color keying.

Fields

Table 1209

Name	Description
keyType	The type of color key: source or destination.
lowerColor	The lower limit of the color key range (included in the range).
upperColor	The upper limit of the color key range (included in the range).
enabled	Determines if the color key is currently enabled, i.e. used in the mixing process.

8.2.15.2.2.12 AutoBlankProperties

Signature

```
typedef struct {
    Bool      enabledOnOutputResolutionChanged;
    Bool      enabledOnFieldRateChanged;
    Bool      enabledOnScanTypeChanged;
    uhColor_t blankColor;
} AutoBlankProperties;
```

Qualifiers
None.

Description
Values of this type represent the auto blanking properties of the mixer. There are a number of events for which the client can instruct the mixer to automatically blank the output. The client can specify a blanking color to be used when automatically blanking, and he can specify on which events to auto-blank.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Fields

Table 1210

Name	Description
`enabledOnOutputResolutionChanged`	If `True`, the platform automatically blanks on an output resolution change.
`enabledOnFieldRateChanged`	If `True`, the platform automatically blanks on an output field rate change.
`enabledOnScanTypeChanged`	If `True`, the platform automatically blanks on an output scan type change.
`blankColor`	The blanking color used by the platform to automatically blank on the above events.

8.2.15.3 Logical Component

8.2.15.3.1 Interface-Role Model

Figure 148 depicts the interface-role model of the Video Mixer. It shows the control and streaming interfaces and the roles the Video Mixer and a Video Mixing Client can have.

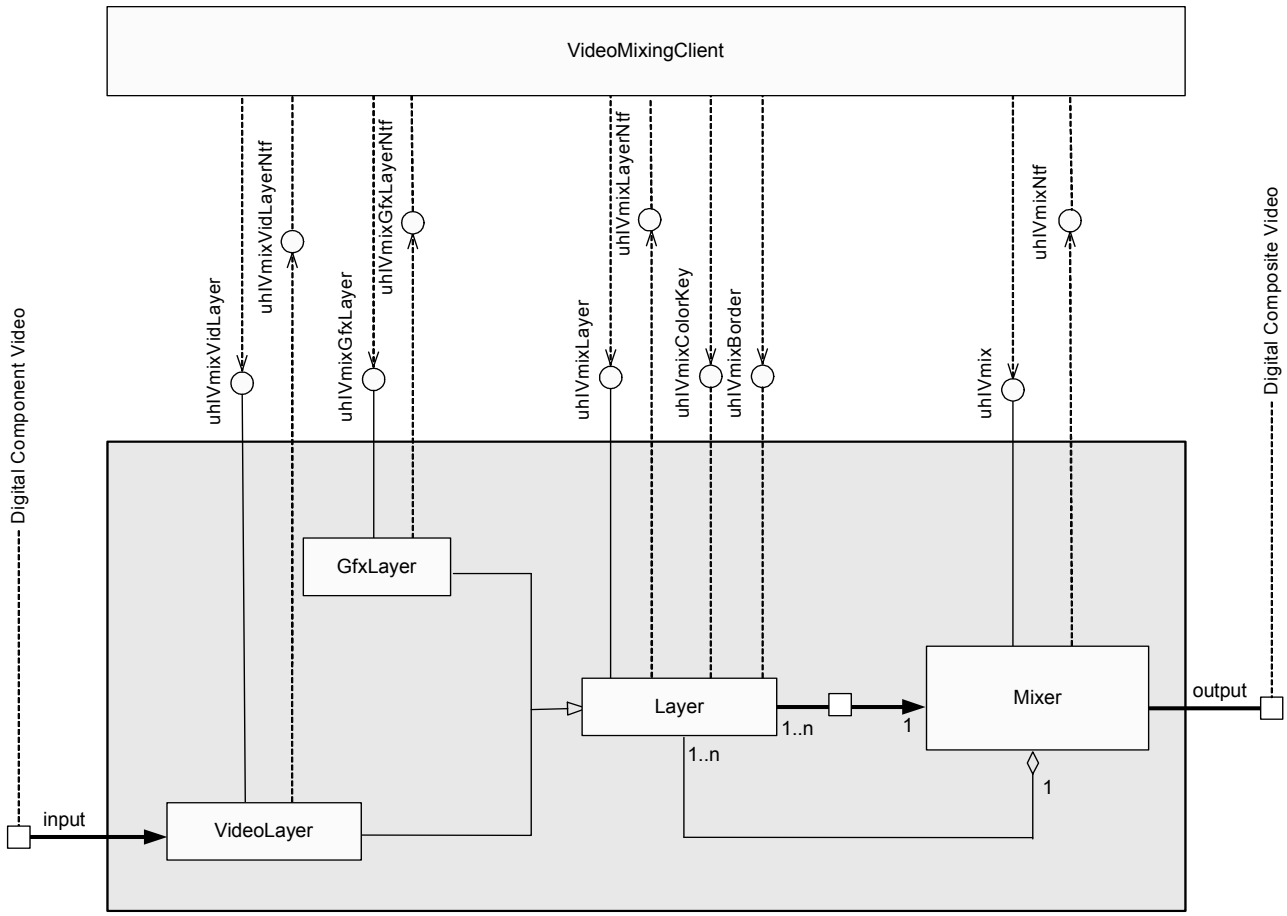


Figure 148 — Interface-Role Model

A VideoMixingClient can interact with two types of layers: a VideoLayer and a

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

GfxLayer. However, these roles have common behavior, which is captured in a general Layer role. Any layer translates its input into an output. For a Video Layer, both the input and output are streams. For a Graphics Layer, the input is a bitmap passed by the client that is translated into an output stream by the layer. In the descriptions of all three layer roles, the streaming behavior is expressed in terms of input and output. However, not all layer roles have been given a stream input and output in Figure 148 in order to keep the figure clean.

In general, a Mixer supports an arbitrary number of layers. In practice, this number is dictated by the hardware. The supported number of layers can be obtained via `uhIVmix`. Navigation over interfaces on different roles is only supported by `uhIVmix`. Via this interface, the `uhIVmixLayer` interface of either a video or graphics layer can be obtained by a VideoMixingClient. The reverse operation is not supported, because this is redundant.

The `uhIVmixVidLayer` and `uhIVmixGfxLayer` interfaces contain the functionality specific to video and graphics layers. Functionality common to all layers is provided on the `uhIVmixLayer` and interface. The `uhIVmix` interface contains functionality that is not related to an individual layer of the mixer but to the whole.

8.2.15.3.2 Diversity

8.2.15.3.2.1 Provided Interfaces

Table 1211

Role	Interface	Presence
`Mixer`	`uhIVmix`	mandatory
`Layer`	`uhIVmixLayer`	mandatory
	`uhIVmixColorKey`	optional
	`uhIVmixBorder`	optional
`VideoLayer`	`uhIVmixVidLayer`	mandatory
`GfxLayer`	`uhIVmixGfxLayer`	mandatory
`VideoMixingClient`	`uhIVmixNtf`	optional
	`uhIVmixLayerNtf`	optional
	`uhIVmixVidLayerNtf`	optional
	`uhIVmixGfxLayerNtf`	optional

8.2.15.3.2.2 Configurable Items

The following table contains a list of role attributes that can be given a value upon creation of a Video Mixer. For a description of the attributes see 8.2.15.4, Roles, and for how they are initialized see subclause 8.2.15.3.3.

Table 1212

Role	Attribute
`Mixer`	`nrLayers`
	`outputProp`
`Layer`	`layerType`
	`bgSupp`
	`minWindowSize`
	`maxLayerSize`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Role	Attribute
	blendRange
	scaleRange
	suppScaleModes
	nrColorKeys
	suppColorKeyTypes
	replaceColor
	panoramaMidScaleRatioRange
	layerEqualsDstWindow
VideoLayer	suppCaptureTypes
	capBufAlignment
GfxLayer	suppFormatTypes
	gfxBufAlignment

Remarks

— The VideoLayer has autonomous unfreeze functionality. Exactly when an autonomous unfreeze happens is platform instance dependent and described in the platform instance document.

8.2.15.3.2.3 Constraints

— `1 <= nrLayers`
— `outputProp` is a supported combination of properties. It is described in the platform instance document which combinations are possible and when the mixer switches from one combination to another.
— `blendRange.min <= blendRange.max`
— `uhVmix_NoScaling` in `suppScaleModes`
— For each *s* in `suppScaleModes`
  — `scaleRange[s].minWidthPercentage <= scaleRange[s].maxWidthPercentage`
  — `scaleRange[s].minHeightPercentage <= scaleRange[s].maxHeightPercentage`
— `nrColorKeys == 0 || suppColorKeyTypes` is not empty
— `panoramaMidScaleRatioRange.min <= panoramaMidScaleRatioRange.max`
— `capBufAlignment >= 1`
— `gfxBufAlignment >= 1`

8.2.15.3.3 Instantiation

8.2.15.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1213

Type	Object	Multiplicity
Mixer	mixer	1
Layer	layer[]	mixer.nrLayers

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
— mixer.blanked == True
— mixer.autoBlanked == False
— mixer.blankColor == black
— mixer.actualBlankColor == black
— mixer.autoBlankProp.enabledOnOutputResolutionChanged == True
— mixer.autoBlankProp.enabledOnFieldRateChanged == True
— mixer.autoBlankProp.enabledOnScanTypeChanged == True
— mixer.autoBlankProp.blankColor == black
— mixer.bgColor == black
— Let nrLayers = mixer.nrLayers
— For each i in {0..nrLayers-1}
   — layer[i].bgColor == black
   — layer[i].nextSettings.layer    ==    (0,    0,    mixer.outputProp.width,
     mixer.outputProp.height)
   — layer[i].nextSettings.dst == layer[i].nextSettings.layer
   — layer[i].nextSettings.scaleMode = uhVmix_NoScaling
   — layer[i].borders.top == 0
   — layer[i].borders.bottom == 0
   — layer[i].borders.left == 0
   — layer[i].borders.right == 0
   — layer[i].borders.hidden == True
   — layer[i].borders.color == black
   — layer[i].targetSettings == layer[i].nextSettings
   — layer[i].activeSettings == layer[i].nextSettings
   — layer[i].startSettingsOnNextInput == False
   — layer[i].hidden == True
   — layer[i].currentBlend == layer[i].blendRange.max
   — layer[i].targetBlend == layer[i].currentBlend
   — For each j in { 0 .. layer[i].nrColorKeys - 1 }
      — layer[i].nextColorKey[j].enabled == False
      — layer[i].activeColorKey[j].enabled == False
   — layer[i].fadingBusy == False
   — layer[i].zoomingBusy == False
— For each i in {0..nrLayers-1}
   — if ( layer[i].layerType == uhVmix_VideoLayer )
      — layer[i].videoLayerStatus == uhVmix_ValStatInvalid
      — layer[i].inputProp.width == maxLayerSize.width
      — layer[i].inputProp.height == maxLayerSize.height
      — layer[i].inputProp.aspectRatioNumerator == 0
      — layer[i].inputProp.aspectRatioDenominator == 1
      — layer[i].inputProp.activeImageArea == (0,0,0,0)
      — layer[i].nextSettings.src    ==    (0,    0,    layer[i].inputProp.width,
        layer[i].inputProp.height)
      — layer[i].blanked == False
      — layer[i].frozen == False
      — layer[i].strobeEnabled == False
      — layer[i].strobeInterval == 0
      — layer[i].captureBusy == False
— For each i in {0..nrLayers-1}
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— if (layer[i].layerType == uhVmix_GfxLayer)
  — For each j in { 0 .. layer[i].nrColorKeys - 1 }
    — layer[i].bufferColorKey[j].enabled == False

Remarks

— The initial values of the color keys of a layer are left unspecified because there is no reasonable default value for them and they are initially disabled anyway.

8.2.15.3.4 Execution Constraints

The GetSuppNrLayers, GetLayerConfig, and GetLayer functions of the uhIVmix interface may be accessed concurrently with respect to each other and all other functions in this interface. For all other functions and interfaces at most one thread may have access to them. Thread-safe functions are the only exception.

8.2.15.4 Roles

8.2.15.4.1 Mixer

Signature

```
role Mixer {
    const UInt32         nrLayers;
    Layer                layer[nrLayers];
    OutputProperties     outputProp;
    Bool                 blanked;
    Bool                 autoBlanked;
    uhColor_t            blankColor;
    AutoBlankProperties autoBlankProp;
    uhColor_t            actualBlankColor;
    uhColor_t            bgColor;
}
```

Qualifiers

— root

Description

A Mixer owns a number of Layers and composes a video stream by mixing these layers, including a virtual layer that provides a background color. The output can be blanked with a specified color (individual layers can also be blanked; see subclause 8.2.15.4.2, Layer). The properties of the output can be queried.

Independent Attributes

Table 1214

Name	Description
nrLayers	The number of layers.
layer	An array of nrLayers layers (layer[0] is bottom layer, layer[nrLayers-1] is top layer). The virtual background layer is not included in this array.
outputProp	The properties of the output window.
blanked	Indicates whether the output window is blanked (by the client) or not.
autoBlanked	Indicates whether the output window is auto blanked (by the platform) or not.
blankColor	The color used for blanking the output window in case of an normal blank.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
autoBlankProp	The current properties of auto blanking, i.e., the blanking color it uses and the events that trigger auto blanking.
actualBlankColor	The color actually used for blanking the output window.
bgColor	The background color for the output window.

Invariants

outputProp is a supported combination of properties (not every combination of values for the fields of type OutputProperties (see subclause 8.2.15.2.2.5) represents a sensible output format).

Instantiation
A Mixer is always instantiated as part of a Video Mixer; see subclause 8.2.15.3.3.

Streaming Behavior
The Mixer receives nrLayers input streams and composes one output stream from these. If blanked == True or autoBlanked == True then the output window shows actualBlankColor. If not, then the output is composed in a number of steps. In each step a layer, beginning with 0 and ending with nrLayers-1, is mixed with the intermediary mixing result (*result*). We start with *result* being the virtual background layer (bgColor). In mixing non-hidden layer *i* with *result*, the pixels in layer[i].activeSettings.layer are mixed with the corresponding pixels in *result*. The output window shows *result* after the top layer has been mixed.

Active Behavior
The Mixer has standard notification behavior; see uhIVmixNtf for the events that are notified.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.4.2 Layer

Signature

```
role Layer {
    const uhVmix_LayerType_t          layerType;
    InputProperties                   inputProp;
    LayerSettings                     nextSettings;
    LayerSettings                     targetSettings;
    LayerSettings                     delayedTargetSettings;
    LayerSettings                     activeSettings;
    Borders                           borders;
    Bool                              startSettingsOnNextInput;
    uhColor_t                         bgColor;
    const Bool                        bgSupp;
    Bool                              hidden;
    const BlendingRange               blendRange;
    UInt32                            currentBlend;
    UInt32                            targetBlend;
    const uhVmix_ScaleModeSet_t       suppScaleModes;
    const ScaleRange                  scaleRange[uhVmix_ScaleMode_t];
    const Bool                        blendingSupp;
    const WindowSize                  minWindowSize;
    const WindowSize                  maxLayerSize;
    const Bool                        layerEqualsDstWindow;
    const UInt32                      nrColorKeys;
    const uhVmix_ColorKeyTypeSet_t    suppColorKeyTypes;
    const uhColor_t                   replaceColor;
    ColorKey                          nextColorKey[nrColorKeys];
    ColorKey                          activeColorKey[nrColorKeys];
    const PanoramaMidScaleRatioRange  panoramaMidScaleRatioRange;
    Bool                              fadingBusy;
    Bool                              zoomingBusy;
}
```

Qualifiers

None.

Description

The `Layer` role provides functionality to get input images into the mixer. `Layer` is an abstract role in the sense that an instance of `Layer` in the platform will always be either an instance of `VideoLayer` or an instance of `GfxLayer`.

Independent Attributes

Table 1215

Name	Description
`layerType`	Indicates if the layer is a video or a graphics layer.
`inputProp`	The input properties of the layer.
`nextSettings`	The layer settings (source window, destination window, layer window, scale mode, and panoramic effect) set by the client. See Figure 149 for an explanation of the relation between nextSettings, targetSettings, and activeSettings.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
targetSettings	The layer settings the layer will move to (smoothly) from the current settings (activeSettings); targetSettings gets the value of nextSettings when the smooth settings of windows begins. See Figure 149 for an explanation of the relation between targetSettings, nextSettings, and activeSettings.
delayedTargetSettings	These are target settings to be used later on. The attribute is used to model the delayed execution of ActivateNewWindowSettings (see uhIVmixLayer_ActivateNewWindowSettings). See also *Active Behavior* of role Layer
activeSettings	The layer settings actually used in the mixing process (during a zoom, for instance, these settings change continually, moving towards targetSettings). See Figure 149 for an explanation of the relation between activeSettings, nextSettings, and targetSettings.
borders	The borders of the layer window (color, thickness of each side, and whether they are visible or not).
startSettingsOnNextInput	If True, a transition from the current settings (activeSettings) to the target settings (targetSettings) will be started on the next input (a new video field/frame in case the layer is a video layer, or a new buffer in case the layer is a graphics layer). Like delayedTargetSettings this attribute is used to model the delayed execution of ActivateNewWindowSettings (see uhIVmixLayer_ActivateNewWindowSettings). See also *Active Behavior* of role Layer.
bgColor	The background color of the layer.
bgSupp	Indicates whether a layer background color is supported or not.
hidden	Indicates if the layer is hidden or used in the mixing process.
blendRange	The blending range of the layer.
currentBlend	The current blending factor (transparency level) of the layer.
targetBlend	The target blending factor when the layer is fading.
suppScaleModes	The set of scale modes supported by the layer.
scaleRange	A mapping assigning a scale range (minimum and maximum scaling factors) to each scale mode (that is supported on this layer).
minWindowSize	The minimum window width and height on this layer.
maxLayerSize	The maximum layer window width and height on this layer.
layerEqualsDstWindow	Indicates if the layer window and the destination window should have the same size and position.
nrColorKeys	The number of color keys supported by the layer.
suppColorKeyTypes	The supported color key types.
replaceColor	The replacement color used for pixels generating a color key hit.
nextColorKey	An array of color keys (as set by the client).
activeColorKey	An array of color keys actually used in the mixing process.
panoramaMidScaleRatioRange	The range of allowed scale ratios applied in the middle of the source window during panoramic scaling.
fadingBusy	Boolean indicating whether fading is performed at this moment or not.
zoomingBusy	Boolean indicating whether zooming is performed at this moment or not.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes

Table 1216

Name	Description
`blendingSupp`	Indicates if blending is supported: `blendingSupp == (blendRange.min != blendRange.max)`.

The following figure exemplifies the role of the attributes `nextSettings`, `targetSettings`, and `activeSettings` in modelling smooth window settings (for the definition of the functions used, see `SetSrcWindow` (subclause 8.2.15.5.3.17), `ActivateNewWindowSettings` (subclause 8.2.15.5.3.26), and `OnNewSettingsCompleted` (subclause 8.2.15.5.4.5)):

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

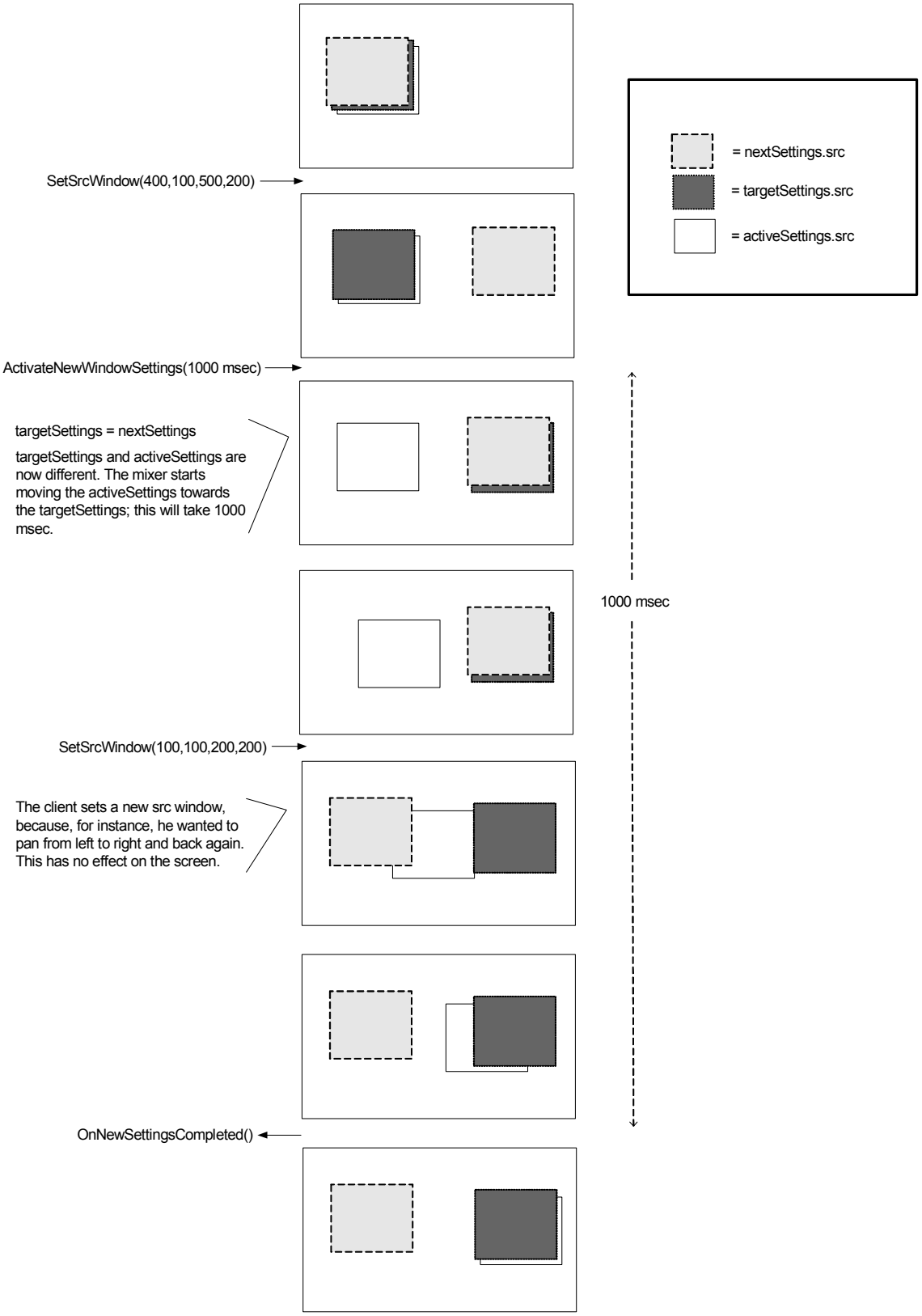


Figure 149 — A "smooth window move" scenario showing the meaning of the different settings-attributes.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Invariants

- `uhVmix_NoScaling` in `suppScaleModes`
- `nextSettings.scaleMode` in `suppScaleModes`
- `blendRange.min <= currentBlend <= blendRange.max`
- `panoramaMidScaleRatioRange.min <= nextSettings.panoramaMidScaleRatio <= panoramaMidScaleRatioRange.max`
- `panoramaMidScaleRatioRange.min <= targetSettings.panoramaMidScaleRatio <= panoramaMidScaleRatioRange.max`
- `panoramaMidScaleRatioRange.min <= activeSettings.panoramaMidScaleRatio <= panoramaMidScaleRatioRange.max`
- For each `i in { 0 .. nrColorKeys-1 }`
  - `nextColorKey[i].keyType` in `suppColorKeyTypes`
  - `activeColorKey[i].keyType` in `suppColorKeyTypes`

Window rules and invariants

The following rules and invariants apply to the source window:

- Its coordinates are specified relative to the input windows upper left corner (==(0,0)) and may be negative.
- It does *not* have to be contained in the input window. It can be positioned freely.
- The width and height of the intersection of source window and input window are at least `minWindowSize.width` and `minWindowSize.height` respectively. If the intersection becomes smaller than the minimum as a result of a change in the input resolution, the platform's behavior is implementation specific. The client should react on such an event.

The following rules and invariants apply to the layer window:

- Its coordinates are specified relative to the output window's left upper corner (==(0,0)) and may be negative.
- It does *not* have to be contained in the output window. It can be positioned freely.
- Its width and height are at least minWindowSize.width and minWindowSize.height respectively, and at most maxLayerSize.width and maxLayerSize.height (the latter because of restrictions on memory usage).

The following rules and invariants apply to the destination window:

- Its coordinates are specified relative to the layer window's upper left corner and may be negative (so, relative to the output window the destination window's upper left coordinate equals *xxxSettings.layer.ul* + *xxxSettings.dst.ul*).
- It does *not* have to be contained in the layer window or output window.
- Its width and height are at least minWindowSize.width and minWindowSize.height respectively.

Instantiation

A Layer is always a VideoLayer or a GfxLayer and instantiated as part of a Video Mixer; see subclause 8.2.15.3.3.

Streaming Behavior

A Layer produces a conceptual output stream of images constructed by cropping, scaling, and positioning the contents present in the input window. These images are continually mixed (see subclause 8.2.15.4.1 Mixer) with those produced by the other layers.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Cropping is done by specifying a source window that selects the whole or a part of the input window. The source window can fall (partly) outside of the input window. The part of the source window that lies outside the input window does not “contain” pixels. Note that the size of the source window, even if much of it is empty, does play a role in scaling.

Scaling is done by specifying a destination window to which the content of the source window is scaled. A scale mode determines how the content of the source window is scaled. The actual scaling factors that are applied must fall in the range scaleRange[scale mode].

If the linear scale mode is applied, constant horizontal and vertical scaling factors are used, and the contents of the source window fill the destination window completely (the parts of the destination window corresponding to empty parts of the source window, show the layer’s background color or, if a background color is not supported, the underlying mixing result).

If the panorama scale mode is applied, a constant vertical scaling factor is used, but the horizontal scaling factor applied to each pixel is determined by a parabola (producing a better picture when showing a 4:3 image on a 16:9 screen, or a 16:9 image on a 4:3 screen).

If the horizontal scale mode is applied only horizontal linear scaling is used. Horizontally, the contents of the source window fill the destination window completely, but vertically no scaling is applied so that the contents of the source window do not completely fill the destination window, or the bottom part of the source window is cropped by the destination window. See Figure 150.

If the no-scaling “scale mode” is applied then also horizontally the contents of the source window might not fill the destination window completely or are cropped by it.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

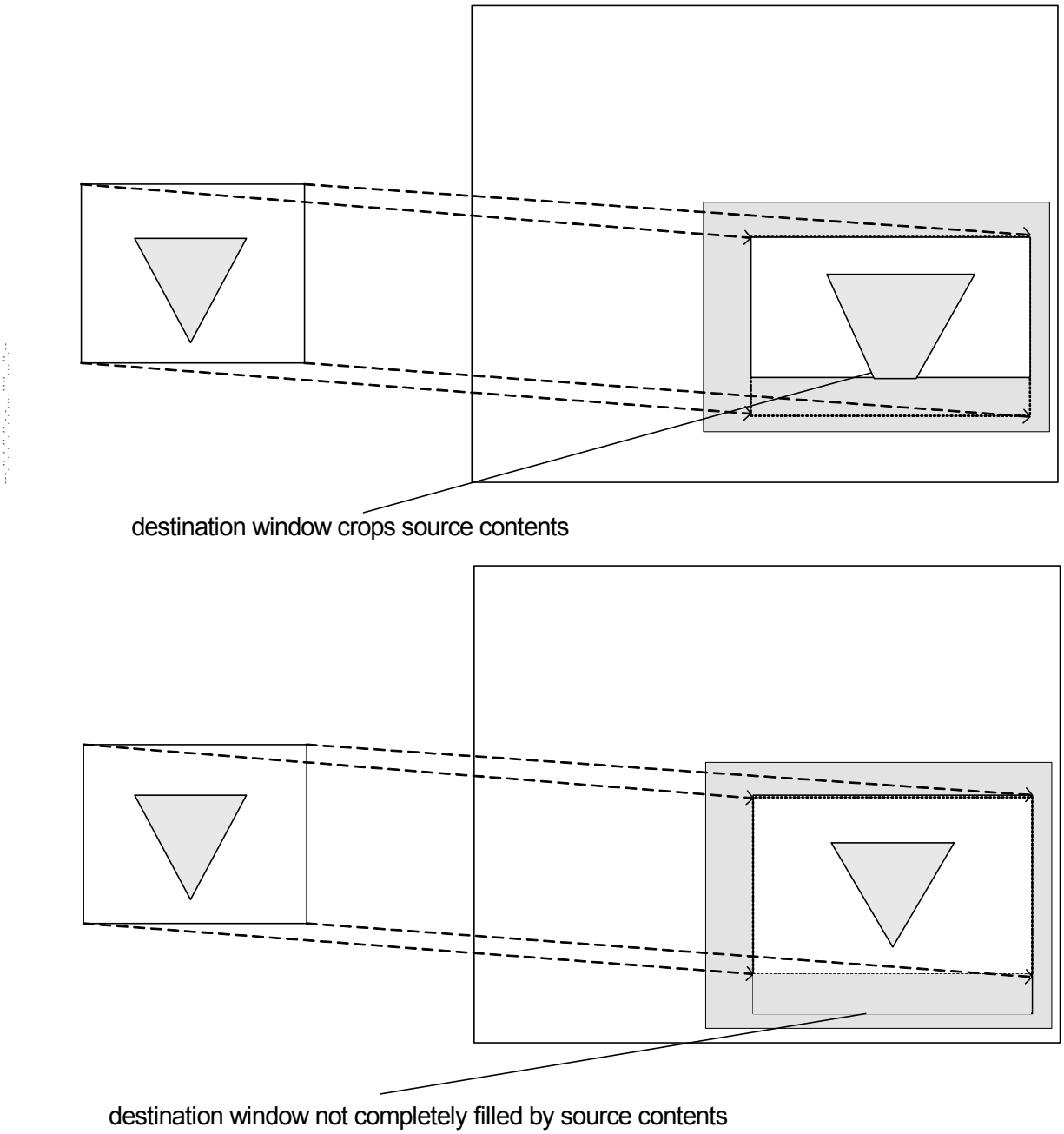


Figure 150 — Scaling source to destination when no vertical and/or horizontal scaling is applied.

Positioning is done by specifying a layer window that is more or less a drawing area positioned somewhere “in” the output window and on which the destination window is positioned. The layer window provides borders and a background color (both optional) and additional cropping (only the part of the destination window within the layer window and in the output window is visible). See Figure 151 for an example.

In addition, a layer can be hidden, meaning that it is not used by the mixer, and a layer can have a blending factor (= transparency level) that tells the mixer how much of the underlying mixing result shines through. The blending factor applies to the entire layer, i.e. the pixels inside the destination window, and the background color pixels outside the destination window (if the layer does not support a background color, blending only applies to the destination window).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

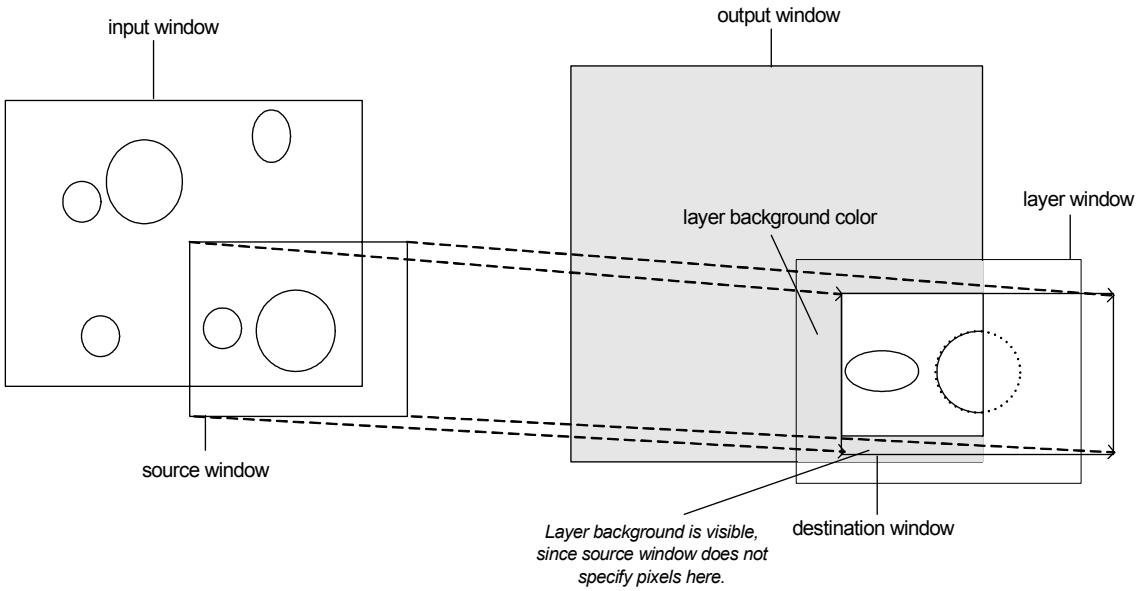


Figure 151 — Cropping, scaling (in Panorama mode), and positioning.

Active Behavior
Two autonomous processes can be active in a layer:

— *Fading*: the client can start a fade from the current blending factor to some specified target blending factor in a specified amount of time. This is done with an asynchronous function (FadeTo, subclause 8.2.15.5.3.10). A fading process is started that will smoothly adapt the layer's blending factor (currentBlend) until the specified target (targetBlend) is reached. The end of fading is notified to the client by means of function `OnFadeCompleted` (subclause 8.2.15.5.4.3). The fading process can be interrupted by calling FadeTo again. A new target blending factor will be set and fading will continue with this new target. The client will be notified of the interruption by means of function `OnFadeInterrupted` (subclause 8.2.15.5.4.2). When fading terminates `currentBlend == targetBlend` will hold.

— *Setting new windows*: the client can set new window coordinates for the different windows of a layer. These new window settings are not effectuated immediately. The client needs to call the function `ActivateNewWindowSettings` (See `uhIVmixLayer_ActivateNewWindowSettings`) and then the settings (nextSettings) are effectuated instantly or smoothly in a specified amount of time (to support smooth zooming). The parameter syncWithNextInput of this function determines if the "zooming" process starts immediately or is delayed until the next input, which is a new video field/frame in case of a video layer, and a new graphics buffer (call to `SetBuffer` (subclause 8.2.15.5.7.7) to be precise) in case of a graphics layer. After the "zooming" process is started, first it will apply the scale mode of `targetSettings` (if no smooth settings of the scale mode is possible) and then it will smoothly adapt the other attributes of `activeSettings` until they reach the values specified in `targetSettings`. The corners of the windows will move at varying speeds to reach their respective destinations at the same time. The borders of the layer window will follow the layer window. Note that the border attributes (like size and color) are not subject to continuous change. The end of zooming is notified to the client by means of function `OnNewSettingsCompleted` (subclause 8.2.15.5.4.5). The zooming process can be interrupted by calling `ActivateNewWindowSettings` again. A new target will be set and zooming will continue with this new target. The process can even be interrupted before it has really started (in case the delayed form of `ActivateNewWindowSettings`, with `syncWithNextInput == True`, was active. The client will be notified of an interruption by means of function `OnNewSettingsInterrupted` (subclause 8.2.15.5.4.4). When zooming terminates `activeSettings == targetSettings` will hold.

The notification behavior of a layer is summarized below (excluding the standard `OnSubscriptionChanged` notifications); see also `uhIVmixGfxLayerNtf`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1217

Event	Action
The layer is currently fading and the client calls `FadeTo` again.	Notify `OnFadeInterrupted(*)`
The layer has completed a fade.	Notify `OnFadeCompleted(*)`
The Layer is currently going to new settings and the client calls `ActivateNewWindowSettings`.	Notify `OnNewSettingsInterrupted(*)`
The Layer has completed applying the new settings.	Notify `OnNewSettingsCompleted(*)`

8.2.15.4.3 VideoLayer

Signature

```
role VideoLayer specializes Layer {
    uhVmix_ValStat_t        videoLayerStatus;
    Bool                    blanked;
    uhColor_t               blankColor;
    Bool                    frozen;
    Bool                    strobeEnabled;
    UInt32                  strobeInterval;
    const uhPixFmtTypeSet_t suppCaptureTypes[uhPixFmtCls_t];
    const uhPixFmtClsSet_t  suppCaptureClasses;
    const UInt32            capBufAlignment;
    Bool                    captureBusy;
}
```

Qualifiers
None.

Description
A VideoLayer is a Layer with video content. It takes its input from another logical component. It provides functionality for blanking, freezing, strobing, and capturing video.

Independent Attributes

Table 1218

Name	Description
`videoLayerStatus`	Validity state of the input properties (`inputProp`: inherited from Layer).
`blanked`	Indicates whether the layer is blanked or not.
`blankColor`	The color used for blanking.
`frozen`	Indicates whether the layer is frozen or not.
`strobeEnabled`	Indicates whether the strobe effect is enabled or disabled.
`strobeInterval`	The minimum number of milliseconds between two updates of the content (as set by the client). The actual interval used is `strobeInterval` rounded up towards a multiple of the duration of a field/frame.
`suppCaptureTypes`	A mapping giving the pixel format types that can be used for the result of capturing, for each pixel format class.
`capBufAlignment`	The byte alignment of a line in a capture buffer. The alignment can pose restrictions on the capture buffers the client can pass to the layer.
`captureBusy`	Boolean indicating whether a capture is performed at this moment or not.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes

Table 1219

Name	Description
`suppCaptureClasses`	The supported pixel format classes for capturing: a class c is in `suppCaptureClasses` if and only if `suppCaptureTypes[c]` is not empty.

Invariants

— `Layer::layerType == uhVmix_VideoLayer`.

Instantiation

A `VideoLayer` is always instantiated as part of a Video Mixer; see subclause 8.2.15.3.3.

Streaming Behavior

— If `blanked == True`, instead of showing the video content in the destination window of the layer, the layer and destination windows show `blankColor` (note that layer borders remain visible, if present).

— If `frozen == True`, then the video content for this layer is not updated.

— if `(strobeEnabled && (strobeInterval > 0))`, then the content is updated every `strobeInterval` (rounded up towards a multiple of the duration of a field/frame) milliseconds, unless `frozen == True`. So, if the strobed video input is frozen, the destination window shows a frozen image, and if the input is unfrozen, we get to see a strobed video stream.

Note that if the layer is blanked, freezing and strobing can still be applied but the result is not visible. They will be the moment the layer is unblanked. Something similar holds for freezing. If the content is frozen, the input can still be strobed but the effect will only be visible when the content is unfrozen (see also in Figure 152).

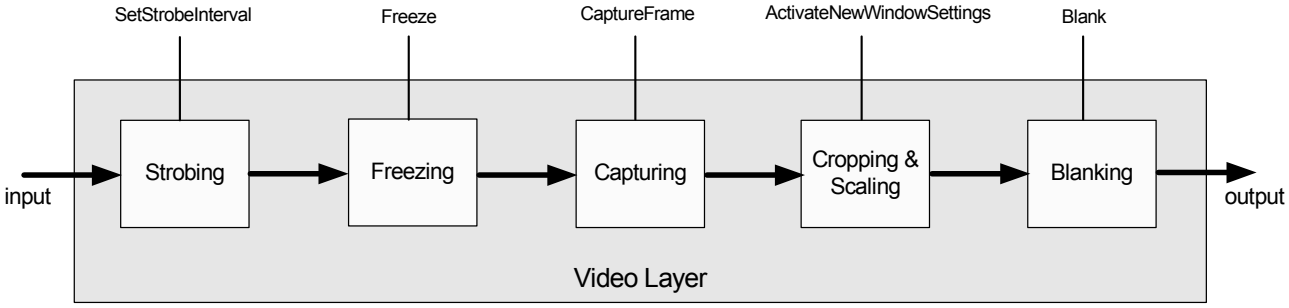


Figure 152 — Dependencies between different blocks of video functionality.

Active Behavior

Besides the active behavior inherited from Layer, a `VidLayer` has standard notification behavior (see `uhIVmixVidLayerNtf` for the notified events) and the following event-driven behavior:

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1220

Event	Action

Processing of a new field/frame starts.	The `VideoLayer` will perform the following actions:
	`Layer::activeColorKey = Layer::nextColorKey`
	— if (`Layer::startSettingsOnNextInput == True`)
	`Layer::startSettingsOnNextInput = False`
	`Layer::targetSettings = delayedTargetSettings`
	— Start setting the windows

8.2.15.4.4 GfxLayer

Signature

```
role GfxLayer specializes Layer {
    GfxBuffer                nextBuffer;
    GfxBuffer                activeBuffer;
    uhColorAlpha_t           nextClut[];
    uhColorAlpha_t           bufferClut[];
    uhColorAlpha_t           activeClut[];
    ColorKey                 bufferColorKey[Layer::nrColorKeys];
    const uhPixFmtTypeSet_t  suppFormatTypes[uhPixFmtCls_t];
    const uhPixFmtClsSet_t   suppFormatClasses;
    const UInt32             gfxBufAlignment;
}
```

Qualifiers
None.

Description
A `GfxLayer` is a Layer with graphics content. It takes its input from the client who provides a buffer with an image. In case the values in this buffer are `CLUT` (Color LookUp Table) values, it will interpret these with a `CLUT` also provided by the client.

Independent Attributes

Table 1221

Name	Description
`nextBuffer`	The pixel buffer containing the input graphics for this layer (as set by the client). See Figure 153 for an explanation of the relation of `nextBuffer` and `activeBuffer`.
`nextClut`	The CLUT used for interpreting the buffer in case it contains CLUT values (as set by the client). See Figure 153 for an explanation of the relation of `nextClut` and `bufferClut`, `activeClut`, `nextBuffer`, and `activeBuffer`.
`activeBuffer`	The pixel buffer actually used in the mixing process; activeBuffer gets the value of `nextBuffer` when a new output composition starts. See Figure 153 for an explanation of the relation of `activeBuffer` and `nextBuffer`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
bufferClut	The CLUT associated with the last buffer passed by the client; bufferClut gets the value of nextClut when the client calls uhIVmixGfxLayer::SetBuffer. See Figure 153 for an explanation of the relation of bufferClut and nextClut, activeClut, nextBuffer, and activeBuffer.
activeClut	The CLUT actually used in the mixing process; activeClut gets the value of bufferClut when a new output composition starts. See Figure 153 for an explanation of the relation of activeClut and nextClut, bufferClut, nextBuffer, and activeBuffer.
bufferColorKey	The color keys associated with the last buffer passed by the client; bufferColorKey gets the value of nextColorKey when the client calls uhIVmixGfxLayer::SetBuffer. bufferColorKey relates to Layer::nextColorKey and Layer::activeColorKey as bufferClut to nextClut and activeClut (See Figure 153).
suppFormatTypes	A mapping giving the pixel format types the client is allowed to use on this graphics layer, for each pixel format class.
gfxBufAlignment	The byte alignment of a line in a graphics buffer. The alignment can pose restrictions on the graphics buffers the client can pass to the layer.

Dependent Attributes

Table 1222

Name	Description
suppFormatClasses	The supported pixel format classes for graphics: a class c is in suppFormatClasses if and only if suppFormatTypes[c] is not empty.

Buffers and CLUTs, which are manipulated independently by the client via the functions SetBuffer and SetClut, are not truelly independent as the contents of a buffer containing CLUT-values have to be interpreted with a specific CLUT. How buffers and CLUTs set with these functions relate to each other is modeled via the different buffer and clut attributes of the GfxLayer role. The following figure depicts a scenario exemplifying the roles of these attributes:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

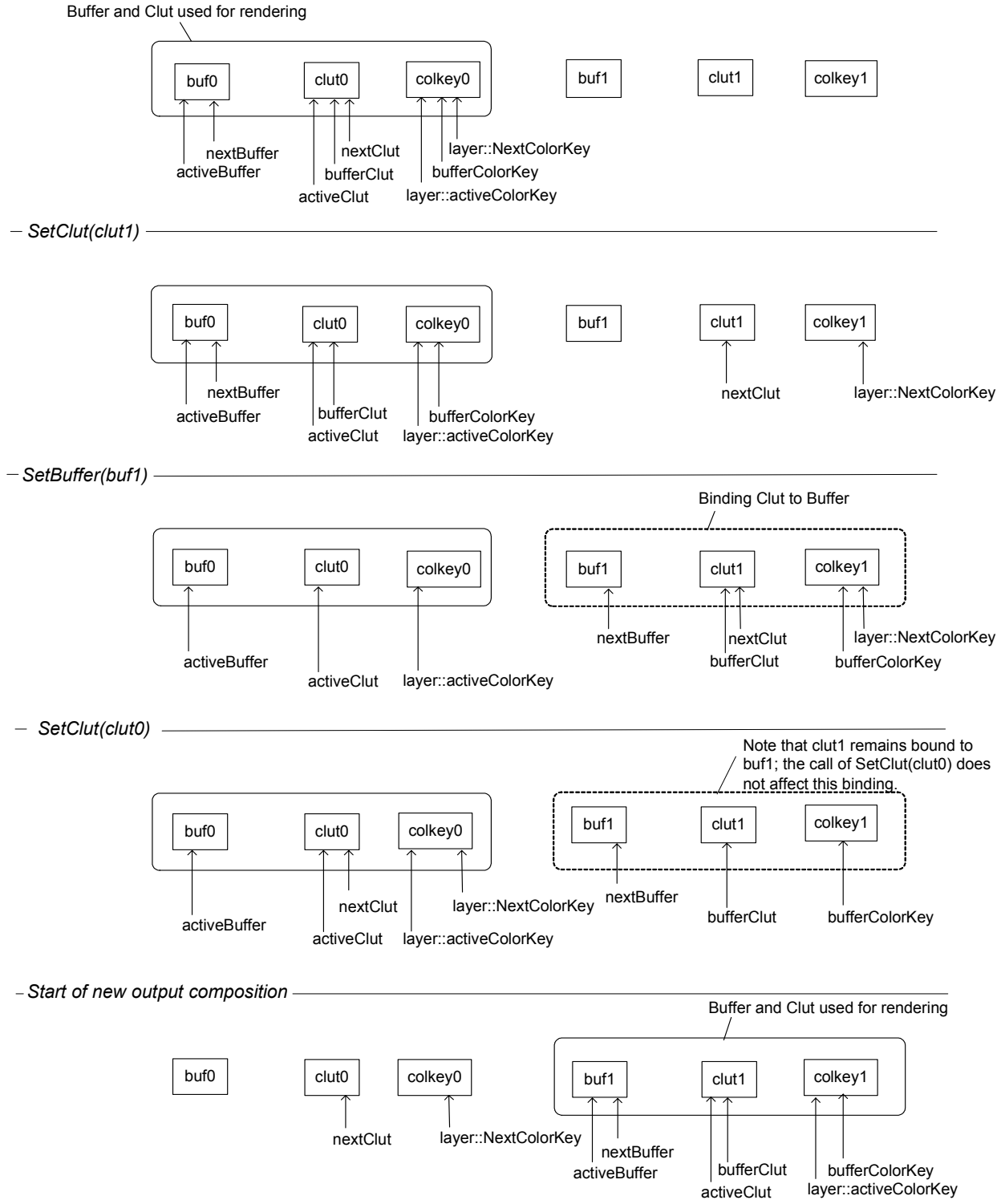


Figure 153 — A scenario showing the meaning of the buffer and clut attributes.

Invariants

— `Layer::layerType == uhVmix_GfxLayer`
— `Layer::inputProp.width == activeBuffer.width`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— `Layer::inputProp.height == activeBuffer.height`
— `Layer::inputProp.aspectRatioNumerator == 0`
— `Layer::inputProp.aspectRatioDenominator == 1`
— `Layer::inputProp.activeImageArea == (0,0,0,0)`

Instantiation
A `GfxLayer` is always instantiated as part of a Video Mixer; see subclause 8.2.15.3.3.

Streaming Behavior
The client can provide the graphics layer with input by passing a buffer and a CLUT. The buffer is used for the next output composition. The last one is used if more than one are passed during the composition of the current output image. The CLUT that is used in a new output composition is the last CLUT that was passed before the last buffer was passed (See Figure 153 for an example). So passing a new CLUT should always be followed by passing a buffer if one wants the CLUT to take effect.

Active Behavior
Besides the active behavior inherited from Layer, a GfxLayer has standard notification behavior (see `uhIVmixGfxLayerNtf` for the notified events) and the following event-driven behavior:

Table 1223

Event	Action

A new output composition starts.	The `GfxLayer` performs the following actions:
	`activeBuffer = nextBuffer`
	`activeClut = bufferClut`
	`Layer::activeColorKey = bufferColorKey`

8.2.15.4.5 VideoMixingClient

Signature
`role` `VideoMixingClient {}`

Qualifiers
— actor

Description
A `VideoMixingClient` represents a user of the interfaces `uhIVmix`, `uhIVmixLayer`, `uhIVmixVidLayer`, and `uhIVmixGfxLayer`.

8.2.15.5 Interfaces

8.2.15.5.1 uhIVmix

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The interface `uhIVmix` provides functionality related to the output and it provides access to the layers. Each layer has a *fixed* identifying number which is determined by the platform. A client can request a layer's `uhIVmixLayer` interface by specifying this number. A layer configuration can be requested that tells the client how many layers are configured in this Video Mixer and whether these are video or graphics layers. It also deals with output stream characteristics (output blanking, querying and setting the output format) and the specification of the color of the virtual background layer.

In the function specifications below, we use the name mixer to denote the instance of the role `Mixer` providing the interface.

Interface ID
`uuid( DCEE9CF7-54B3-46d9-86B3-7E328D4FDF73 )`

8.2.15.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIVmixNtf                 *pINotify,
    [in] UInt32                     cookie,
    [in] uhVmix_Notifications_t     notifs );
```

Qualifiers
— subscribe-function

8.2.15.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIVmixNtf                 *pINotify,
    [in] UInt32                     cookie,
    [in] uhVmix_Notifications_t     notifs );
```

Qualifiers
— unsubscribe-function

8.2.15.5.1.3 GetSuppNrLayers

Signature
```
uhErrorCode_t GetSuppNrLayers ( [out] pUInt32 pNrLayers );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of layers supported by the video mixer.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1224

Name	Description
pNrLayers	Output parameter returning the supported number of layers.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pNrLayers == mixer.nrLayers

Remarks

— The number of layers will at least be 1.

8.2.15.5.1.4 GetLayerConfig

Signature

```
uhErrorCode_t GetLayerConfig (
    [in] UInt32                                 nrLayers,
    [out,size_is(nrLayers)] uhVmix_LayerType_t  *pLayerType );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the types of all the layers. See subclause 8.2.15.2.1.3

uhVmix_LayerType_t for an enumeration of the different layer types. Note that layer 0 is the bottom layer and layer nrLayers-1 is the top layer (layers are identified by their position in the z-order).

Parameters

Table 1225

Name	Description
nrLayers	The supported number of layers for this mixer (as returned by the function GetSuppNrLayers).
pLayerType	Output parameter returning an array with each layer's type.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`nrLayers == mixer.nrLayers`

Action
None.

Postcondition
— for all `i, 0 <= i < nrLayers`:

`pLayerType[i] == mixer.layer[i].layerType`

8.2.15.5.1.5 GetLayer

Signature
```
uhErrorCode_t GetLayer ( [in] UInt32 layerNr, [out] uhIVmixLayer **ppLayer );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the `uhIVmixLayer` interface of layer layerNr.

Parameters

Table 1226

Name	Description
`layerNr`	The layer number.
`ppLayer`	Output parameter returning a pointer to the `uhIVmixLayer` interface pointer of layer *layerNr* to be returned.

Return Values
Standard.

Precondition
`0 <= layerNr < mixer.nrLayers`

Action
None.

Postcondition
— *ppLayer contains a pointer to the `uhIVmixLayer` interface of `mixer.layer[layerNr]`.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.1.6 GetOutputProperties

Signature

```
uhErrorCode_t GetOutputProperties (
    [out] puhVmix_ValStat_t     pStatus,
    [out] pUInt32               pWidth,
    [out] pUInt32               pHeight,
    [out] puhVmix_FieldRate_t   pFieldRate,
    [out] puhVmix_ScanType_t    pScanType );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the output properties of the video mixer: the number of pixels per line and the number of lines of the output window, the field rate of the output, and the scan type.

Parameters

Table 1227

Name	Description
`pStatus`	Output parameter that indicates the validity of the width and height parameters.
`pWidth`	Output parameter returning the number of pixels per line of the output window.
`pHeight`	Output parameter returning the number of lines of the output window.
`pFieldRate`	Output parameter returning the field rate of the output (see subclause 8.2.15.2.1.18).
`pScanType`	Output parameter returning the scan type of the output (see subclause 8.2.15.2.1.19).

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*pStatus == mixer.outputProp.status`
— `*pWidth == mixer.outputProp.width`
— `*pHeight == mixer.outputProp.height`
— `*pFieldRate == mixer.outputProp.fieldRate`
— `*pScanType == mixer.outputProp.scanType`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.1.7 BlankOutput

Signature

```
uhErrorCode_t BlankOutput (
    [in] Bool      blank,
    [in] uhColor_t blankColor );
```

Qualifiers

— synchronous

— single-threaded

Description

Blanks or unblanks the output of the video mixer. The blanking of the video mixer is independent of the blanking of individual layers. When the output is blanked the specified (RGB) blanking color is shown full screen. The parameter *blankColor* has no effect in case of an unblank (except that `GetOutputBlanked` will return this value).

Parameters

Table 1228

Name	Description
`blank`	Indicates whether the output has to be blanked (`True`) or unblanked (`False`).
`blankColor`	The blanking color.

Return Values

Standard.

Precondition

`True`

Action

— if `(blank == True)`
  — Blank the screen with `blankColor` (on the next field).
  — Modify `mixer.blanked, mixer.blankColor,mixer.actualBlankColor.`
— if `(blank == False)`
  — Modify `mixer.blanked, mixer.blankColor.`
  — Unblank the screen (on the next field).

Postcondition

— `mixer.blanked == blank.`
— `mixer.blankColor == blankColor.`
— if `(blank == True)`
  — `mixer.actualBlankColor == blankColor.`

Remarks

— This function is synchronous and lasts up to the next field to complete (when the screen is actually (un)blanked).

— Note that the screen is only unblanked if no auto blanking is currently active (see Figure 154).

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.1.8 GetOutputBlanked

Signature

```
uhErrorCode_t GetOutputBlanked (
    [out] pBool      pBlanked,
    [out] puhColor_t pBlankColor  );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether the output of the mixer is blanked or not, and the (RGB) blanking color that was set.

Parameters

Table 1229

Name	Description
`pBlanked`	Output parameter that indicates whether the output is blanked (`True`) or not (`False`).
`pBlankColor`	Output parameter returning the blanking color.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*pBlanked == mixer.blanked`

— `*pBlankColor == mixer.blankColor`

Remarks

— If the output is unblanked this function still returns the blanking color that was passed although it was not meaningful.

8.2.15.5.1.9 EnableAutoBlank

Signature

```
uhErrorCode_t EnableAutoBlank (
    [in] Bool      enableOnOutputResolutionChanged,
    [in] Bool      enableOnFieldRateChanged,
    [in] Bool      enableOnScanTypeChanged,
    [in] uhColor_t blankColor );
```

Qualifiers

— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— single-threaded

Description
Enables or disables auto blanking. The client can specify on which events auto blanking has to take place. Auto blanking of the video mixer is independent of the blanking of individual layers, but not completely independent from normal mixer blanking (see Figure 154).

	Normal blanking	Auto blanking

blank	BlankOutput(True, blankColor)
	{
	actualBlankColor = blankColor;
	blanked = True;
	}
		EnableAutoBlank(event1, event2, event3, blankColor)
		{
		if (!blanked)
		actualBlankColor = /*auto*/ blankColor;
		autoBlanked = True;
		}

unblank	BlankOutput(False, blankColor)
	{
	blanked = False;
	}
		AutoBlankUnBlank( )
		{
		autoBlanked = False;
		}

If (blanked || autoBlanked) *the screen shows* actualBlankColor

event1: OnOutputResolutionChanged
event2: OnFrameRateChanged
event3: OnScanTypeChanged

Figure 154 — The combined behavior of normal blanking and auto blanking in pseudo-code.

Parameters

Table 1230

Name	Description
`enableOnOutputResolutionChanged`	Indicates whether the output has to be blanked (`True`) or unblanked (`False`) when the output resolution changes.
`enableOnFieldRateChanged`	Indicates whether the output has to be blanked (`True`) or unblanked (`False`) when the output's field rate changes.
`enableOnScanTypeChanged`	Indicates whether the output has to be blanked (`True`) or unblanked (`False`) when the output's scan type changes.
`blankColor`	The blanking color to be used by auto blanking.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Modify `mixer.autoBlankProp`

Postcondition
— `mixer.autoBlankProp.enabledOnOutputResolutionChanged == enableOnOutputResolutionChanged.`
— `mixer.autoBlankProp.enabledOnFieldRateChanged == enableOnFieldRateChanged.`
— `mixer.autoBlankProp.enabledOnScanTypeChanged == enableOnScanTypeChanged.`
— `mixer.autoBlankProp.blankColor == blankColor.`

8.2.15.5.1.10 GetAutoBlankEnabled

Signature
```
uhErrorCode_t GetAutoBlankEnabled (
    [out] pBool      pBlankOnOutputResolutionChangedEnabled,
    [out] pBool      pBlankOnFieldRateChangedEnabled,
    [out] pBool      pBlankOnScanTypeChangedEnabled,
    [out] puhColor_t pBlankColor  );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the auto blank settings of the mixer, i.e. for which mixer events auto blanking is enabled and the (RGB) blanking color that is used.

Parameters

Table 1231

Name	Description
`pBlankOnOutputResolutionChangedEnabled`	Output parameter that indicates whether an auto blank is performed (`True`) or not (`False`) on an output resolution change.
`pBlankOnFieldRateChangedEnabled`	Output parameter that indicates whether an auto blank is performed (`True`) or not (`False`) on an output field rate change.
`pBlankOnScanTypeChangedEnabled`	Output parameter that indicates whether an auto blank is performed (`True`) or not (`False`) on an output scan type change.
`pBlankColor`	Output parameter returning the blanking color used in auto blanking.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— `*pBlankOnOutputResolutionChangedEnabled == mixer.autoBlankProp.enabledOnOutputResolutionChanged`
— `*pBlankOnFieldRateChangedEnabled == mixer.autoBlankProp.enabledOnFieldRateChanged`
— `*pBlankOnScanTypeChangedEnabled == mixer.autoBlankProp.enabledOnScanTypeChanged`
— `*pBlankColor == mixer.autoBlankProp.blankColor`

8.2.15.5.1.11 AutoBlankUnblank

Signature
```
uhErrorCode_t AutoBlankUnblank ( );
```

Qualifiers
— synchronous

— single-threaded

Description
Deactivates the auto blank (if it is active). See also Figure 154.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Action
— `if (mixer.autoBlanked == True)`
  — `Modify mixer.autoBlanked, mixer.actualBlankColor.`

Postcondition
— `mixer.autoBlanked == False.`
— `mixer.actualBlankColor == blankColor`

Remarks
— This function may return before the screen is actually unblanked. The mixer may keep the screen blanked a little longer, if this is required to generate a stable output.

— Note that the screen is only unblanked if no normal client blanking is currently active (see Figure 154).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.1.12 SetBgColor

Signature
```
uhErrorCode_t SetBgColor ( [in] uhColor_t bgColor );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the (RGB) color of the background layer.

Parameters

Table 1232

Name	Description
`bgColor`	The color for the background layer.

Return Values
Standard.

Precondition
`True`

Action
— modify `mixer.bgColor`
— Request the mixer to set its background color to `bgColor` (on the next field).

Postcondition
`mixer.bgColor == bgColor`

8.2.15.5.1.13 GetBgColor

Signature
```
uhErrorCode_t GetBgColor ( [out] puhColor_t pBgColor );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the (RGB) color value of the background layer.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1233

Name	Description
pBgColor	Output parameter returning the background color.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pBgColor == mixer.bgColor

8.2.15.5.2 uhIVmixNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a Mixer. The Subscribe function of uhIVmix can be used to subscribe to these events.

Interface ID
uuid(F6F22B78-C4B3-44b7-8C38-DA5D6FFE5FF5)

8.2.15.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVmix_Notifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.15.5.2.2 OnNewProperties

Signature
```
[async] Void OnNewProperties (
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32                     cookie,
    [in] Bool                       outputResolutionChanged,
    [in] UInt32                     outputWidth,
    [in] UInt32                     outputHeight,
    [in] Bool                       fieldRateChanged,
    [in] uhVmix_FieldRate_t         fieldRate,
    [in] Bool                       scanTypeChanged,
    [in] uhVmix_ScanType_t          scanType,
    [in] UInt32                     nrVidLayers,
    [in, size_is(nrVidLayers)]
        uhVmix_NewVidLayerProp_t    *pVidLayerProps,
    [in] UInt32                     nrGfxLayers,
    [in, size_is(nrGfxLayers)]
        uhVmix_NewGfxLayerProp_t    *pGfxLayerProps );
```

Qualifiers
None.

Description
This function is called by a Mixer whenever the input properties of a layer or the output properties of the mixer have changed. It is possible that this function is called even if none of these properties have changed, in case a platform instance defines other events that trigger this function (e.g.: a channel change).

Parameters

Table 1234

Name	Description
`cookie`	The cookie identifying the subscription that enabled this notification.
`outputResolutionChanged`	Indicates whether the output resolution has changed (`True`) or not (`False`).
`outputWidth`	The width of the output window.
`outputHeight`	The height of the output window.
`fieldRateChanged`	Indicates whether the output field rate has changed (`True`) or not (`False`).
`fieldRate`	The field rate of the output (see subclause 8.2.15.2.1.18).
`scanTypeChanged`	Indicates whether the output scan type has changed (`True`) or not (`False`).
`scanType`	The scan type of the output (see subclause 8.2.15.2.1.19).
`nrVidLayers`	The number of video layers of the mixer.
`pVidLayerProps`	An array containing the properties of the video layers (plus indications which properties have changed). The layers are sorted in increasing order (first video layer in this array is the one with the lowest layer number).
`nrGfxLayers`	The number of graphics layers of the mixer.
`pGfxLayerProps`	An array containing the properties (resolutions) of the graphics layers. The layers are sorted in increasing order (first graphics layer in this array is the one with the lowest layer number).

Return Values
None.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
— Any callback-compliant action.

Postcondition
`True`

8.2.15.5.3 uhIVmixLayer

Qualifiers
None.

Description
The interface `uhIVmixLayer` represents a single layer in the mixer model. The interface deals with the layer's background color, hiding, blending, fading, windows, and scaling.

In the function specifications below, we use the name layer to denote the instance of the role Layer providing the interface.

Interface ID
`uuid( 59756FEE-8E9A-46b6-865A-1A4C24FF222E )`

8.2.15.5.3.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIVmixLayerNtf               *pINotify,
    [in] UInt32                        cookie,
    [in] uhVmix_LayerNotifications_t   notifs );
```

Qualifiers
— subscribe-function

8.2.15.5.3.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIVmixLayerNtf               *pINotify,
    [in] UInt32                        cookie,
    [in] uhVmix_LayerNotifications_t   notifs );
```

Qualifiers
— unsubscribe-function

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.3.3 GetBgSupp

Signature
```
uhErrorCode_t GetBgSupp ( [out]  pBool pBgSupp );
```

Qualifiers
— synchronous

— thread-safe

Description
Reports whether a background color is supported for this layer or not.

Parameters

Table 1235

Name	Description
pBgSupp	Output parameter returning whether a background color is supported or not.

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
*pBgSupp == layer.bgSupp
```

8.2.15.5.3.4 SetBgColor

Signature
```
uhErrorCode_t SetBgColor ( [in] uhColor_t bgColor );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the (RGB) background color for this layer.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1236

Name	Description
`bgColor`	The layer background color to be set.

Return Values
Standard.

Precondition
`layer.bgSupp == True`

Action
— Modify `layer.bgColor`.
— Request the layer to set its background color to *bgColor* (on the next field).

Postcondition
— `layer.bgColor == bgColor`.

Remarks
— There is a default background color for all layers that support a background color configured at startup time.

— `bgColor` will be converted to the pixel format currently used for this layer.

8.2.15.5.3.5 GetBgColor

Signature
```
uhErrorCode_t GetBgColor ( [out] puhColor_t pBgColor );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the (RGB) background color for this layer.

Parameters

Table 1237

Name	Description
`pBgColor`	Output parameter returning the layer background color.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`layer.bgSupp == True`

Action
None.

Postcondition
`*pBgColor == layer.bgColor`

Remarks
Although the RGB color value set with `SetBgColor` is converted to the pixel format used on this layer, the original value that was set is returned when the background color is queried.

8.2.15.5.3.6 Hide

Signature
```
uhErrorCode_t Hide ( [in] Bool hide );
```

Qualifiers
— synchronous

— single-threaded

Description
Hides or shows the layer on which this interface operates. When hidden, the layer is no longer used in the mixing process, therefore no video/graphics, no borders, and no background color of this layer will be visible.

Parameters

Table 1238

Name	Description
`hide`	Indicates whether the layer has to be hidden (`True`) or shown (`False`).

Return Values
Standard.

Precondition
`True`

Action
— if `(hide == True)`
  — Hide the layer (on the next field).
  — Modify `layer.hidden`.
— if `(hide == False)`
  — Modify `layer.hidden`.
  — Unhide the layer (on the next field).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`layer.hidden == hide.`

Remarks
This function is synchronous and lasts up to the next field to complete (when the layer is actually hidden/shown).

8.2.15.5.3.7 GetHidden

Signature
```
uhErrorCode_t GetHidden ( [out] pBool pHidden );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the layer is hidden or not.

Parameters

Table 1239

Name	Description
`pHidden`	Output parameter returning whether the layer is hidden (`True`) or not (`False`).

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pHidden == layer.hidden`

8.2.15.5.3.8 GetBlendFactorRange

Signature
```
uhErrorCode_t GetBlendFactorRange (
    [out] pUInt32        pMinBlendFactor,
    [out] pUInt32        pMaxBlendFactor );
```

Qualifiers
— synchronous

— thread-safe

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the blending factor range of the layer on which this interface operates. When the minimum blending factor equals the maximum blending factor, blending is not supported. Otherwise, the minimum corresponds to fully transparent and the maximum to fully opaque.

Parameters

Table 1240

Name	Description
`pMinBlendFactor`	Output parameter returning the minimum blending factor of this layer.
`pMaxBlendFactor`	Output parameter returning the maximum blending factor of this layer.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
```
*pMinBlendFactor == layer.blendRange.min

*pMaxBlendFactor == layer.blendRange.max
```

8.2.15.5.3.9 GetBlendFactor

Signature
```
uhErrorCode_t GetBlendFactor ( [out] pUInt32 pBlendFactor );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns the layer-blending factor of the layer as set by the client (with `FadeTo`).

Parameters

Table 1241

Name	Description
`pBlendFactor`	Output parameter returning the blending factor (transparency level) of this layer as set by the client.

Return Values
Only non-standard error values are listed:

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1242

Name	Description
UH_ERR_VMIX_FADING	The layer is currently fading. This is just a warning that the blending factor that is returned may not have been reached yet.

Precondition
`layer.blendingSupp == True`

Action
None.

Postcondition
`*pBlendFactor == layer.targetBlend`

8.2.15.5.3.10 FadeTo

Signature
```
uhErrorCode_t FadeTo ( [in] UInt32 blendFactor, [in] UInt32 nrMilliseconds );
```

Qualifiers
- asynchronous
- single-threaded

Description
Fades the layer from the current blending factor to the specified target blending factor in a specified amount of time. If one just wants to apply a new blending factor this function can be called with `nrMilliseconds == 0`. The fade does not apply to the borders of the layer window.

Parameters

Table 1243

Name	Description
blendFactor	The blending factor that is the target of the fade.
nrMilliseconds	The time the fade should take in milliseconds.

Return Values
Standard.

Precondition
`layer.blendingSupp == True`

Action
- Modify `layer.targetBlend`
- if `(layer.fadingBusy == True)`
  - Notify `OnFadeInterrupted`
- else
  - Modify `layer.fadingBusy`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Request layer to start fading.

Postcondition
— `layer.targetBlend == blendFactor`
— `layer.fadingBusy == True`

Asynchronous Action
— Modify `layer.currentBlend`.
— if `(layer.targetBlend == layer.currentBlend)`
  — Modify `layer.fadingBusy`
  — Notify `OnFadeCompleted`

Asynchronous Postcondition
— `layer.currentBlend == blendFactor`
— `layer fadingBusy == False`

Remarks
— See also Active Behavior of Layer (subclause 8.2.15.4.2).

— In general calling this function with `nrMilliseconds != 0` only makes sense when at least 16 blending steps are supported, otherwise the fade may be perceived as jerky.

8.2.15.5.3.11 GetMinWindowSize

Signature
```
uhErrorCode_t GetMinWindowSize (
    [out]  pUInt32           pMinWidth,
    [out]  pUInt32           pMinHeight );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum window width and height on this layer. This minimum applies to all the layer's windows, plus the intersection of the input window and the source window.

Parameters

Table 1244

Name	Description
`pMinWidth`	Output parameter returning the minimum window width.
`pMinHeight`	Output parameter returning the minimum window height.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— *pMinWidth == layer.minWindowSize.width
— *pMinHeight == layer.minWindowSize.height

8.2.15.5.3.12 GetMaxLayerWindowSize

Signature
```
uhErrorCode_t GetMaxLayerWindowSize (
    [out]  pUInt32          pMaxWidth,
    [out]  pUInt32          pMaxHeight );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum layer window width and height on this layer.

Parameters

Table 1245

Name	Description
pMaxWidth	Output parameter returning the maximum layer window width
pMaxHeight	Output parameter returning the maximum layer window height.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
— *pMaxWidth == layer.maxLayerSize.width
— *pMaxHeight == layer.minLayerSize.height

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.3.13 SetLayerWindow

Signature
```
uhErrorCode_t SetLayerWindow ( [in] uhRect_t layerRect );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the layer window of the layer on which this interface operates. Note that in order to let the mixer start using the new layer window, the client must call `ActivateNewWindowSettings` (see subclause 8.2.15.5.3.26).

Parameters

Table 1246

Name	Description
`layerRect`	the coordinates of the layer window relative to the output window.

Return Values
Standard.

Precondition
— `layerRect.lr.x - layerRect.ul.x >= layer.minWindowSize.width`
— `layerRect.lr.y - layerRect.ul.y >= layer.minWindowSize.height`
— `layerRect.lr.x - layerRect.ul.x <= layer.maxLayerSize.width`
— `layerRect.lr.y - layerRect.ul.y <= layer.maxLayerSize.height`

Action
— Modify `layer.nextSettings.layer`

Postcondition
`layer.nextSettings.layer == layerRect`

Remarks
Not all implementations support both a layer window and a destination window. See `LayerEqualsDstWindow` in 8.2.15.5.3.27.

8.2.15.5.3.14 GetLayerWindow

Signature
```
uhErrorCode_t GetLayerWindow ( [out] puhRect_t pLayerRect );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the target coordinate values of the layer window of the layer. The target coordinate values always equal the actual coordinates, except during a smooth transition of windows.

Parameters

Table 1247

Name	Description
`pLayerRect`	Output parameter returning the target coordinates of the layer window.

Return Values
Only non-standard error values are listed:

Table 1248

Name	Description
`UH_ERR_VMIX_SETTING_WINDOWS`	The layer is currently settings its windows. This is just a warning that the window that is returned may not have been reached yet.

Precondition
`True`

Action
None.

Postcondition
`*pLayerRect == layer.targetSettings.layer`

8.2.15.5.3.15 SetDstWindow

Signature
```
uhErrorCode_t SetDstWindow ( [in] uhRect_t dstRect );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the destination window of the layer on which this interface operates. Note that in order to let the mixer start using the new destination window, the client must call `ActivateNewWindowSettings` (see subclause 8.2.15.5.3.26).

Parameters

Table 1249

Name	Description
`dstRect`	The coordinates of the destination window (relative to the layer window).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
— `dstRect.lr.x - dstRect.ul.x >= layer.minWindowSize.width`
— `dstRect.lr.y - dstRect.ul.y >= layer.minWindowSize.height`

Action
— Modify `layer.nextSettings.dst`

Postcondition
`layer.nextSettings.dst == dstRect`

Remarks
Not all implementations support both a layer window and a destination window. See `LayerEqualsDstWindow` in 8.2.15.5.3.27.

8.2.15.5.3.16 GetDstWindow

Signature
```
uhErrorCode_t GetDstWindow ( [out] puhRect_t pDstRect );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the target coordinates of the destination window of the layer. The target coordinate values always equal the actual coordinates, except during a smooth transition of windows.

Parameters

Table 1250

Name	Description
`pDstRect`	Output parameter returning the target coordinates of the destination window.

Return Values
Only non-standard error values are listed:

Table 1251

Name	Description
`UH_ERR_VMIX_SETTING_WINDOWS`	The layer is currently settings its windows. This is just a warning that the window that is returned may not have been reached yet.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pDstRect == layer.targetSettings.dst
```

8.2.15.5.3.17 SetSrcWindow

Signature
```
uhErrorCode_t SetSrcWindow ( [in] uhRect_t srcRect );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the source window of the layer on which this interface operates. Note that in order to let the mixer start using the new source window, the client must call `ActivateNewWindowSettings` (see subclause 8.2.15.5.3.26).

Parameters

Table 1252

Name	Description
`srcRect`	The coordinates of the source window (relative to the input window).

Return Values
Standard.

Precondition
— `srcRect.lr.x - srcRect.ul.x >= layer.minWindowSize.width`
— `srcRect.lr.y - srcRect.ul.y >= layer.minWindowSize.height`

Action
— modify `layer.nextSettings.src`

Postcondition
```
layer.nextSettings.src == srcRect
```

8.2.15.5.3.18 GetSrcWindow

Signature
```
uhErrorCode_t GetSrcWindow ( [out] puhRect_t pSrcRect );
```

Qualifiers
— synchronous

— thread-safe

-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the target coordinates of the source window of the layer. The target coordinate values always equal the actual coordinates, except during a smooth transition of windows.

Parameters

Table 1253

Name	Description
`pSrcRect`	Output parameter returning the target coordinates of the source window.

Return Values
Only non-standard error values are listed:

Table 1254

Name	Description
`UH_ERR_VMIX_SETTING_WINDOWS`	The layer is currently settings its windows. This is just a warning that the window that is returned may not have been reached yet.

Precondition
`True`

Action
None.

Postcondition
`*pSrcRect == layer.targetSettings.src`

8.2.15.5.3.19 GetSuppScaleModes

Signature
```
uhErrorCode_t GetSuppScaleModes ( [out] puhVmix_ScaleModeSet_t pScaleModes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the scale modes supported by the layer on which this interface operates. The returned integer value represents the OR-ed supported scale modes (see subclause 8.2.15.2.1.16 uhVmix_ScaleMode_t).

Parameters

Table 1255

Name	Description
`pScaleModes`	Output parameter returning the supported scale modes.

Return Values
Standard.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition
`True`

Action
None.

Postcondition
`*pScaleModes == layer.suppScaleModes.`

8.2.15.5.3.20 GetScaleRange

Signature
```
uhErrorCode_t GetScaleRange (
    [in] uhVmix_ScaleMode_t         scaleMode,
    [out] pUInt32                   pMinWidthPercentage,
    [out] pUInt32                   pMinHeightPercentage,
    [out] pUInt32                   pMaxWidthPercentage,
    [out] pUInt32                   pMaxHeightPercentage );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns, for a specified scale mode, the minimum and maximum, horizontal and vertical scale factors in percentages (see WindowSize subclause 8.2.15.2.2.7). For the uhVmix_NoScaling scale mode, for instance, all output parameters should return 100 (meaning 100%).

Parameters

Table 1256

Name	Description
`scaleMode`	The scale mode of which the scaling range is queried.
`pMinWidthPercentage`	Output parameter returning the minimum horizontal scaling factor.
`pMinHeightPercentage`	Output parameter returning the minimum vertical scaling factor.
`pMaxWidthPercentage`	Output parameter returning the maximum horizontal scaling factor.
`pMaxHeightPercentage`	Output parameter returning the maximum vertical scaling factor.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
- `*pMinWidthPercentage == layer.scaleRange[scaleMode].minWidthPercentage`
- `*pMinHeightPercentage == layer.scaleRange[scaleMode].minHeightPercentage`
- `*pMaxWidthPercentage == layer.scaleRange[scaleMode].maxWidthPercentage`
- `*pMaxHeightPercentage == layer.scaleRange[scaleMode].maxHeightPercentage`

8.2.15.5.3.21 SetScaleMode

Signature
```
uhErrorCode_t SetScaleMode ( [in] uhVmix_ScaleMode_t scaleMode );
```

Qualifiers
- synchronous
- single-threaded

Description
Sets one of the supported scale modes of the layer on which this interface operates. The scale mode defines how the contents of the source window have to be scaled to the destination window. Note that in order to let the mixer start using the new scale mode, the client must call `ActivateNewWindowSettings` (see subclause 8.2.15.5.3.26).

Parameters

Table 1257

Name	Description
`scaleMode`	The scale mode.

Return Values
Standard.

Precondition
`scaleMode in layer.suppScaleModes`

Action
- Modify `layer.nextSettings.scaleMode`

Postcondition
`layer.nextSettings.scaleMode == scaleMode`

8.2.15.5.3.22 GetScaleMode

Signature
```
uhErrorCode_t GetScaleMode ( [out] puhVmix_ScaleMode_t pScaleMode );
```

Qualifiers
- synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Returns the target scale mode of the layer. The target scale mode always equals the actual scale mode, except during a smooth transition of windows.

Parameters

Table 1258

Name	Description
`pScaleMode`	Output parameter returning the target scale mode for this layer.

Return Values
Only non-standard error values are listed:

Table 1259

Name	Description
`UH_ERR_VMIX_SETTING_WINDOWS`	The layer is currently settings its windows. This is just a warning that the scale mode that is returned may not have been set yet.

Precondition
`True`

Action
None.

Postcondition
`*pScaleMode == layer.targetSettings.scaleMode`

8.2.15.5.3.23 GetPanoramaMidScaleRange

Signature
```
uhErrorCode_t GetPanoramaMidScaleRange (
    [out] pUInt32            pMinMidScaleRatio,
    [out] pUInt32            pMaxMidScaleRatio );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the minimum and maximum scale ratio that may be set for the middle of the source window (in panoramic scaling).

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1260

Name	Description
`pMinMidScaleRatio`	Output parameter returning the minimum scale ratio (in percentages) that may be set for the middle of the source window (in panoramic scaling).
`pMaxMidScaleRatio`	Output parameter returning the maximum scale ratio (in percentages) that may be set for the middle of the source window (in panoramic scaling).

Return Values
Standard.

Precondition
`uhVmix_Panorama in layer.suppScaleModes`

Action
None.

Postcondition
— `*pMinMidScaleRatio == layer.panoramaMidScaleRatioRange.min`
— `*pMaxMidScaleRatio == layer.panoramaMidScaleRatioRange.max`

8.2.15.5.3.24 SetPanoramaMidScale

Signature
```
uhErrorCode_t SetPanoramaMidScale ( [in] UInt32 midScaleRatio );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the scale ratio that has to be used in the middle of the source window, in case panoramic scaling is applied. The mid scale ratio that is passed via this function, together with the source and destination windows completely defines the parabola that is used to create the panoramic effect. Figure 155 below shows the part of the parabola used to scale the right half of the source window. The parabola *f* defines a scale factor for all pixels in the right half of the source window (numbered *0* to *src.width / 2*). The mean scale factor applied to a pixel must equal *dest.width / src.width*, otherwise the scaled pixels of the source window do not completely fit in, or fill, the destination window.

Note that in order to let the mixer start using the panoramic scale ratio that is set, the client must also set the panoramic scale mode and call `ActivateNewWindowSettings` (see subclause 8.2.15.5.3.26).

The mid scale ratio is the ratio of the mid scale factor and mean scale factor in percentages. In other words the mid scale ratio is defined as mid scale factor / mean scale factor * 100, see Figure 156.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

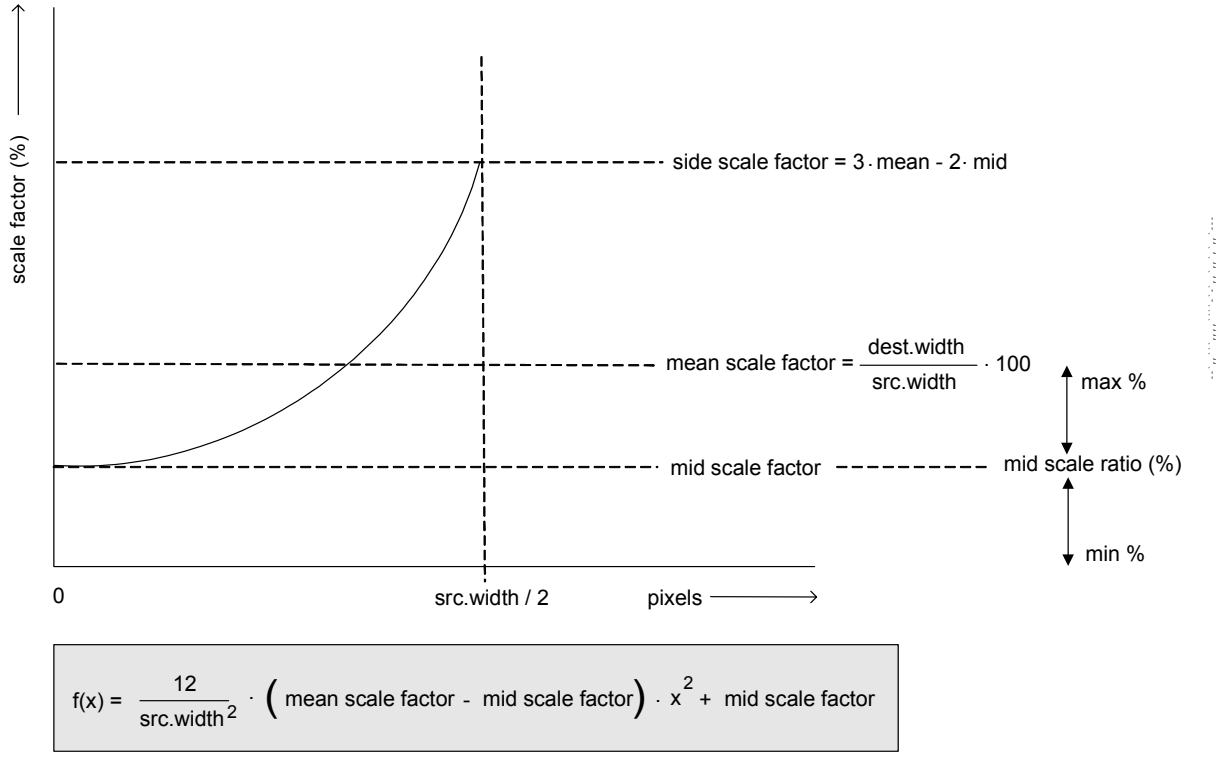


Figure 155 — The scaling parabola f(x) defined by dest.width, src.width, and the mid scale factor.

$$\text{mid scale ratio} = \frac{\text{mid scale factor}}{\text{mean scale factor}} \cdot 100$$

Figure 156 — Definition of mid scale ratio

Parameters

Table 1261

Name	Description
`midScaleRatio`	The scale ratio that is applied to the middle of the source window (when panoramic scaling is enabled) in percentages.

Return Values
Standard.

Precondition
— `uhVmix_Panorama` in `layer.suppScaleModes`
— `layer.panoramaMidScaleRatioRange.min` <= `midScaleRatio` <= `layer.panoramaMidScaleRatioRange.max`

Action
— Modify `layer.nextSettings.panoramaMidScaleRatio`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
`layer.nextSettings.panoramaMidScaleRatio == midScaleRatio`

8.2.15.5.3.25 GetPanoramaMidScale

Signature
```
uhErrorCode_t GetPanoramaMidScale ( [out] pUInt32 pMidScaleRatio );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the mid scale ratio that, together with the source and destination windows, determines the panoramic effect (see Figure 155 for details). The ratio that was set as target is returned, which always equals the actual ratio that is used, except during a smooth transition of windows.

Parameters

Table 1262

Name	Description
`pMidScaleRatio`	Output parameter returning the scale ratio (in percentages) that is applied to the middle of the source window (when panoramic scaling is enabled).

Return Values
Standard.

Precondition
`uhVmix_Panorama` in layer`.suppScaleModes`

Action
None.

Postcondition
`*pMidScaleRatio == layer.targetSettings.panoramaMidScaleRatio`

8.2.15.5.3.26 ActivateNewWindowSettings

Signature
```
uhErrorCode_t ActivateNewWindowSettings (
    [in] Bool       syncWithNextInput,
    [in] UInt32     nrMilliSeconds );
```

Qualifiers
— asynchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The values last set with `SetLayerWindow`, `SetDstWindow`, `SetSrcWindow`, `SetScaleMode` and possibly `SetPanoramaMidScale` are effectuated in a specified amount of time. If one just wants to set these values instantly, this function can be called with `nrMilliseconds == 0`.

The windows will change smoothly, each one arriving at its destination at the same time. If panoramic scaling is used, the parabola that creates the panoramic effect will also be changed smoothly (the mid scale factor changes linearly, just like the window coordinates). If another scale mode is selected, it is applied instantly before the smooth transitions start.

Parameters

Table 1263

Name	Description
`syncWithNextInput`	Wait for new input (`True`) or do it now (`False`).
`nrMilliSeconds`	The number of milliseconds that the smooth setting of windows should take.

Return Values
Standard.

Precondition
- Width of intersection of `layer.nextSettings.src` and `input window` >= `layer.minWindowSize.width`
- Height of intersection of `layer.nextSettings.src` and `input window` >= `layer.minWindowSize.height`
- if (`layer.nextSettings.scaleMode` in `{uhVmix_Linear, uhVmix_Panorama}`)
  - the widths and heights of `layer.nextSettings.src` and `layer.nextSettings.dst` are allowed according to `layer.scaleRange` and `layer.nextSettings.scaleMode`
- if (`layer.nextSettings.scaleMode` in `{uhVmix_Horizontal}`)
  - the widths of `layer.nextSettings.src` and `layer.nextSettings.dst` are allowed according to `layer.scaleRange` and `layer.nextSettings.scaleMode`
- `if (layer.layerEqualsDstWindow == True)`
  - `layer.nextSettings.dst.ul.x == 0`
  - `layer.nextSettings.dst.ul.y == 0`
  - `layer.nextSettings.dst.lr.x == (layer.nextSettings.layer.lr.x - layer.nextSettings.layer.ul.x)`
  - `layer.nextSettings.dst.lr.y == (layer.nextSettings.layer.lr.y - layer.nextSettings.layer.ul.y)`

Action
- Modify `layer.startSettingsOnNextInput`
- if (`syncWithNextInput == False`)
  - Modify `layer.targetSettings`
- else
  - Modify `layer.delayedTargetSettings`
- if (`layer.zoomingBusy == True`)
  - Notify `OnNewSettingsInterrupted`
- else
  - Modify `layer.zoomingBusy`
  - if (`syncWithNextInput == False`)
    - Request layer to start setting the windows.
  - else

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— See *Active Behavior* in subclause 8.2.15.4.3 for video and subclause 8.2.15.5.7.7 for graphics.

Postcondition
— `layer.startSettingsOnNextInput == syncWithNextInput`
— if (`syncWithNextInput == False`)
  — `layer.targetSettings == layer.nextSettings`
— else
  — `layer.delayedTargetSettings == layer.nextSettings`
  — `layer.zoomingBusy == True`

Asynchronous Action
— Modify `layer.activeSettings`
— if (`layer.targetSettings == layer.activeSettings`)
  — Modify `layer.zoomingBusy`
  — Notify `OnNewSettingsCompleted`

Asynchronous Postcondition
— `layer.targetSettings == layer.activeSettings`
— `layer.zoomingBusy == False`

Remarks
— See also Active Behavior of Layer (subclause 8.2.15.4.2).

— If the layer is a video layer then `syncWithNextInput` means: wait for the next field/frame. If the layer is a graphics layer it means: wait for the next buffer (= call to `SetBuffer`); this way resetting the windows and the graphics buffer can be done atomically.

— Settings (of the windows, the scale mode, or the panoramic effect) done after calling but before completion of this function don't have an effect on the current call.

8.2.15.5.3.27 LayerEqualsDstWindow

Signature
```
uhErrorCode_t LayerEqualsDstWindow ( [out] pBool pEqual );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the layer window and the destination window should have equal size and position.

Parameters

Table 1264

Name	Description
`pEqual`	Output parameter returning whether the layer window and the destination window should have equal size and position (`True`) or not (`False`).

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pEqual == layer.layerEqualsDstWindow`

Remarks
— An implementation is allowed to support the layer and the destination window as being the same. Such a case requires the client to give layer- and destination window equal size and position.

— See the pre-condition in ActivateNewWindowSettings in 8.2.15.5.3.26

8.2.15.5.4 uhIVmixLayerNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a Layer. The Subscribe function of `uhIVmixLayer` can be used to subscribe to these events.

Interface ID
`uuid( 4E050424-7A90-4189-8743-D7F160B02002 )`

8.2.15.5.4.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVmix_LayerNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.15.5.4.2 OnFadeInterrupted

Signature
```
[async] Void OnFadeInterrupted ( [in] UInt32 cookie );
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers
None.

Description
This function is called by a Layer whenever an unfinished fade is interrupted (by another call to `FadeTo`); see Active Behavior of Layer (subclause 8.2.15.4.2).

Parameters

Table 1265

Name	Description
`cookie`	The cookie identifying the subscription that enabled this notification.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
Each call of `FadeTo` results in either a `OnFadeInterrupted` or a `OnFadeCompleted` notification.

8.2.15.5.4.3 OnFadeCompleted

Signature
```
[async] Void OnFadeCompleted ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by a Layer whenever a fade is completed; see Active Behavior of Layer (subclause 8.2.15.4.2).

Parameters

Table 1266

Name	Description
`cookie`	The cookie identifying the subscription that enabled this notification.

-`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
— Each call of `FadeTo` results in either a `OnFadeInterrupted` or a `OnFadeCompleted` notification.

8.2.15.5.4.4 OnNewSettingsInterrupted

Signature
```
[async] Void OnNewSettingsInterrupted ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by a Layer whenever a smooth setting of windows has been interrupted (by another call to `ActivateNewWindowSettings`); see Active Behavior of Layer (subclause 8.2.15.4.2).

Parameters

Table 1267

Name	Description
`cookie`	The cookie identifying the subscription that enabled this notification.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks
Each call of `ActivateNewWindowSettings` results in either a `OnNewSettingsInterrupted` or `OnNewSettingsCompleted` notification.

8.2.15.5.4.5 OnNewSettingsCompleted

Signature
```
[async] Void OnNewSettingsCompleted ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by a Layer whenever a smooth setting of windows has been completed; see Active Behavior of Layer (subclause 8.2.15.4.2).

Parameters

Table 1268

Name	Description
`cookie`	The cookie identifying the subscription that enabled this notification.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
Each call of `ActivateNewWindowSettings` results in either a `OnNewSettingsInterrupted` or `OnNewSettingsCompleted` notification.

8.2.15.5.5 uhlVmixVidLayer

Qualifiers
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The interface `uhIVmixVidLayer` is meant to control a video layer. It adds video specific functionality to the layer functionality already provided by the `uhIVmixLayer` interface, i.e. blanking, freezing, strobing, getting the input properties, and capturing frames.

In the function specifications below, we use the name layer to denote the instance of the role `VideoLayer` providing the interface.

Interface ID
`uuid( 7FBD4059-2EF7-47d6-ABD9-0CA78A9A104F )`

8.2.15.5.5.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIVmixVidLayerNtf             *pINotify,
    [in] UInt32                          cookie,
    [in] uhVmix_VidLayerNotifications_t  notifs );
```

Qualifiers
— subscribe-function

8.2.15.5.5.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIVmixVidLayerNtf             *pINotify,
    [in] UInt32                          cookie,
    [in] uhVmix_VidLayerNotifications_t  notifs );
```

Qualifiers
— unsubscribe-function

8.2.15.5.5.3 Blank

Signature
```
uhErrorCode_t Blank (
    [in] Bool      blank,
    [in] uhColor_t blankColor );
```

Qualifiers
— synchronous

— single-threaded

Description
Blanks or unblanks the layer window on which this interface operates. When a video layer is blanked, then the part of the layer window within the borders of the layer will show the specified (RGB) blanking color.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1269

Name	Description
`blank`	Indicates whether the video layer has to be blanked (`True`) or unblanked (`False`).
`blankColor`	The blanking color.

Return Values
Standard.

Precondition
`True`

Action
- if (`blank == True`)
  - Blank the layer with `blankColor` (on the next field).
  - Modify `layer.blanked, layer.blankColor.`
- if (`blank == False`)
  - Modify `layer.blanked, layer.blankColor.`
  - Unblank the layer (on the next field).

Postcondition
- `layer.blanked == blank.`
- `layer.blankColor == blankColor.`

Remarks
This function is synchronous and lasts up to the next field to complete (when the layer is actually (un)blanked).

8.2.15.5.5.4 GetBlanked

Signature
```
uhErrorCode_t GetBlanked (
    [out] pBool      pBlanked,
    [out] puhColor_t pBlankColor );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns whether this layer is currently blanked or not. It also returns the (last set) blanking color (in RGB).

Parameters

Table 1270

Name	Description
`pBlanked`	Output parameter indicating whether the video layer is blanked (`True`) or not (`False`).
`pBlankColor`	Output parameter returning the current blanking color.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pBlanked == layer.blanked`
— `*pBlankColor == layer.blankColor`

8.2.15.5.5.5 Freeze

Signature
```
uhErrorCode_t Freeze ( [in] Bool freeze );
```

Qualifiers
— synchronous

— single-threaded

Description
Freezes or unfreezes the video content of this video layer. If frozen, the video content of this layer is not updated until the layer is unfrozen.

Parameters

Table 1271

Name	Description
`freeze`	Indicates whether the layer should be frozen (`True`) or unfrozen (`False`).

Return Values
Standard.

Precondition
`True`

Action
— Modify `layer.frozen`
— Request the video layer to (un)freeze (on the next field).

Postcondition
`layer.frozen == freeze.`

Remarks

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— Freeze on a frozen layer and unfreeze on an unfrozen layer are void operations.

8.2.15.5.5.6 GetFrozen

Signature
```
uhErrorCode_t GetFrozen ( [out] pBool pFrozen );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether this layer is currently frozen or not.

Parameters

Table 1272

Name	Description
`pFrozen`	Output parameter indicating whether the video layer is frozen (`True`) or not (`False`).

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pFrozen == layer.frozen.`

8.2.15.5.5.7 EnableStrobe

Signature
```
uhErrorCode_t EnableStrobe ( [in] Bool enable );
```

Qualifiers
— synchronous

Description
Enables or disables the strobe effect.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1273

Name	Description
`enable`	Boolean to enable (`True`) or disable (`False`) the strobe effect.

Return Values
Standard.

Precondition
`True`

Action
— Modify `layer.strobeEnabled`

Postcondition
— `layer.strobeEnabled == enable`

8.2.15.5.5.8 GetStrobeEnabled

Signature
```
uhErrorCode_t GetStrobeEnabled ( [out] pBool pEnabled );
```

Qualifiers
— synchronous

Description
Returns whether the strobe effect is enabled or not.

Parameters

Table 1274

Name	Description
`pBool`	Output parameter returning whether the strobe effect is enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pEnabled == layer.strobeEnabled`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.5.9 SetStrobeInterval

Signature
```
uhErrorCode_t SetStrobeInterval ( [in] UInt32 strobeInterval );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the time interval between two updates of this video layer. The interval can be an arbitrary number of milliseconds, but the actual interval depends on the field rate of the outgoing video. After the strobe interval has expired, a new layer update is done on the next field interrupt and a new interval starts at that time.

Example: `SetStrobeInterval(45)` on an outgoing video field rate of 50Hz will effectively be the same as `SetStrobeInterval(60)` - after 40 ms 2 fields have been skipped and to output a new field we will at least have to wait till 60 ms have passed (instead of 45 ms).

Parameters

Table 1275

Name	Description
`strobeInterval`	The time interval in milliseconds between two layer updates.

Return Values
Standard.

Precondition
`True`

Action
— Modify `layer.strobeInterval`
— Request the video layer to apply `strobeInterval` (on the next field).

Postcondition
`layer.strobeInterval == strobeInterval`

Remarks
Strobe interval values in between 0 and 20 (in case of 50 Hz frame rate) will virtually disable the feature, use the `EnableStrobe()` function to enable/disable the strobe effect.

8.2.15.5.5.10 GetStrobeInterval

Signature
```
uhErrorCode_t GetStrobeInterval ( [out] pUInt32 pStrobeInterval );
```

Qualifiers
— synchronous

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Returns the strobe interval: the number of milliseconds between two updates of this video layer. It returns the strobe interval as it was set (not the actual amount of time between to updates).

Parameters

Table 1276

Name	Description
`pStrobeInterval`	Output parameter returning the strobe interval.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pStrobeInterval == layer.strobeInterval`

8.2.15.5.5.11 GetInputResolution

Signature
```
uhErrorCode_t GetInputResolution (
    [out] puhVmix_ValStat_t                pStatus,
    [out] pUInt32                          pWidth,
    [out] pUInt32                          pHeight );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the resolution of the input of this layer (number of pixels per line, and number of lines).

Parameters

Table 1277

Name	Description
`pStatus`	Output parameter that indicates the validity of the width and height parameters.
`pWidth`	Output parameter returning the number of pixels per line of the input window.
`pHeight`	Output parameter returning the number of lines of the input window.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pStatus == layer.videoLayerStatus`
— `*pWidth == layer.inputProp.width`
— `*pHeight == layer.inputProp.height`

8.2.15.5.5.12 GetInputAspectRatio

Signature
```
uhErrorCode_t GetInputAspectRatio (
    [out] puhVmix_ValStat_t             pStatus,
    [out] pUInt16                       pNumerator,
    [out] pUInt16                       pDenominator );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the aspect ratio a:b of the input of this layer (a is the numerator and b is the denominator). If the input signal does not contain information on the aspect ratio, then a == 0 and b == 1.

Parameters

Table 1278

Name	Description
`pStatus`	Output parameter that indicates the validity of the numerator and denominator parameters.
`pNumerator`	Output parameter returning the numerator of the aspect ratio of the input.
`pDenominator`	Output parameter returning the denominator of the aspect ratio of the input.

Return Values
Standard.

Precondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— `*pStatus == layer.videoLayerStatus`
— `*pNumerator == layer.inputProp.aspectRatioNumerator`
— `*pDenominator == layer.inputProp.aspectRatioDenominator`

8.2.15.5.5.13 GetInputActiveImageArea

Signature
```
uhErrorCode_t GetInputActiveImageArea (
    [out] puhVmix_ValStat_t                pStatus,
    [out] puhRect_t                        pActiveImageArea );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the active image area of the input of this layer. The active image area is the part of the input video that contains the actual video (excluding black bars). If the input signal does not contain information on the active image area, then this function will return (0,0,0,0).

Parameters

Table 1279

Name	Description
`pStatus`	Output parameter that indicates the validity of the active image area parameter.
`pActiveImageArea`	Output parameter returning the active image area of the input.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pStatus == layer.videoLayerStatus`
— `*pActiveImageArea == layer.inputProp.activeImageArea`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.5.14 GetSuppCaptureFormatClasses

Signature

```
uhErrorCode_t GetSuppCaptureFormatClasses (
    [out] puhPixFmtClsSet_t       pFormatClasses );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the supported format classes for video capturing. The returned integer value represents the OR-ed supported format classes.

Parameters

Table 1280

Name	Description
`pFormatClasses`	Output parameter returning the supported format classes (for capturing).

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

```
*pFormatClasses == layer.suppCaptureClasses
```

8.2.15.5.5.15 GetSuppCaptureFormatTypes

Signature

```
uhErrorCode_t GetSuppCaptureFormatTypes (
    [in] uhPixFmtCls_t            formatClass,
    [out] puhPixFmtTypeSet_t      pFormatTypes );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the supported format types of the specified pixel format class for video capturing. The returned integer value represents the OR-ed supported format types.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1281

Name	Description
`formatClass`	The pixel class for which the supported pixel format types are requested.
`pFormatTypes`	Output parameter returning the supported pixel format types for the pixel format class `formatClass`.

Return Values
Standard.

Precondition
— `formatClass` in `layer.suppCaptureClasses`

Action
None.

Postcondition
`*pFormatTypes == layer.suppCaptureTypes[formatClass]`

8.2.15.5.5.16 GetCapBufAlignment

Signature
```
uhErrorCode_t GetCapBufAlignment ( [out] pUInt32  pAlignment );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the byte alignment required of capture buffer lines. The byte alignment poses restrictions on the stride the client specifies for capture buffers (see `CaptureFrame`).

Parameters

Table 1282

Name	Description
`pAlignment`	The byte alignment of the lines of a capture buffer.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
```
*pAlignment == layer.capBufAlignment
```

8.2.15.5.5.17 CaptureFrame

Signature
```
uhErrorCode_t CaptureFrame (
    [in] uhRect_t                          captureRect,
    [in] UInt32                            width,
    [in] UInt32                            height,
    [in] UInt32                            stride,
    [in] uhPixFmtCls_t                     formatClass,
    [in] uhPixFmtType_t                    formatType,
    [in] UInt32                            bufferSize,
    [out, size_is(bufferSize)] pUInt8      pBuffer);
```

Qualifiers
- asynchronous
- single-threaded

Description
Returns a screen-shot of this video layer. The content of the intersection of the input window and the specified capture window is captured (the client should position this capture window in the input window, but as the input window can change autonomously, containment of the capture window can not be guaranteed). If the layer is frozen, then this function will capture from the frozen content (see Figure 152). The window content is scaled in width and height as specified by the client. The client allocates a buffer and sees to it that its size is at least *stride* * *height* (see Figure 157 for an explanation of *stride*). There is no support for pixel formats that require multiple buffers (e.g. one for Y, one for UV). `CaptureFrame` always captures a video frame whether the input is interlaced or not.

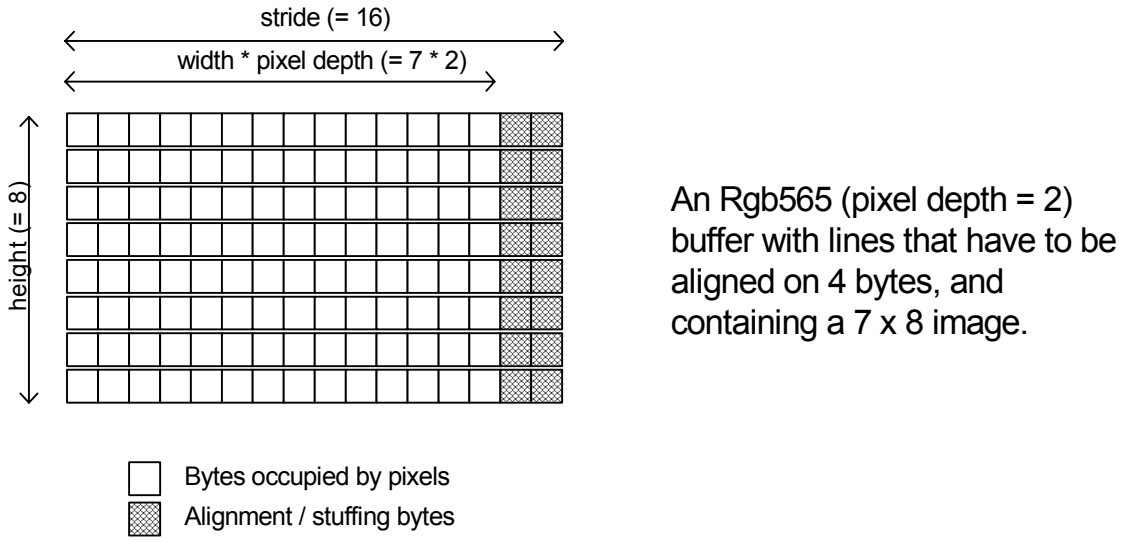


Figure 157 — An example showing the relation between pixel type, image size, buffer size and stride.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1283

Name	Description
captureRect	The rectangular part of the input window that is captured.
width	The number of pixels per line the captured image should be.
height	The number of lines the captured image should be.
stride	The stride (in bytes) of the provided buffer. See Figure 157.
formatClass	The requested pixel format class.
formatType	The requested pixel format type.
bufferSize	The size of the buffer in bytes.
pBuffer	Pointer to the first element of the buffer where the video is to be captured.

Return Values
Standard.

Precondition
— `layer.captureBusy == False`
— `stride >= width * pixel depth of formatType`
— `stride mod layer.capBufAlignment == 0`
— `bufferSize == stride * height`
— `formatClass` in `layer.suppCaptureClasses`, see function `GetSuppFormatClasses`.
— `formatType` in `layer.suppCaptureTypes[formatClass]`, see function `GetSuppFormatTypes`.

Action
— Modify `layer.captureBusy`

Postcondition
`layer.captureBusy == True`

Asynchronous Action
— Modify `layer.captureBusy`
— Notify `OnFrameCaptured`

Asynchronous Postcondition
— `layer.captureBusy == False`
— `pBuffer` points to a screen-shot of the captured video of this layer.

8.2.15.5.6 uhIVmixVidLayerNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in a `VidLayer`. The *Subscribe* function of `uhIVmixVidLayer` can be used to subscribe to these events.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 0D064434-3B48-4954-B846-AF97C591BC8A )`

8.2.15.5.6.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhVmix_VidLayerNotifications_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.15.5.6.2 OnAutoUnfrozen

Signature
```
[async] Void OnAutoUnfrozen ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called by a `VidLayer` whenever the layer becomes autonomously unfrozen, see *Active Behavior* of VideoLayer (subclause 8.2.15.4.3). When exactly this happens is platform instance dependent and described in the platform instance document.

Parameters

Table 1284

Name	Description
`cookie`	The cookie identifying the subscription that enabled this notification.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.6.3 OnFrameCaptured

Signature
```
[async] Void OnFrameCaptured ( [in] UInt32 cookie );
```

Qualifiers
None.

Description
This function is called whenever capturing a frame is finished.

Parameters

Table 1285

Name	Description
cookie	The cookie identifying the subscription that enabled this notification.

Return Values
None.

Precondition
```
True
```

Action
Any callback-compliant action.

Postcondition
```
True
```

8.2.15.5.7 uhlVmixGfxLayer

Qualifiers
None.

Description
A graphics layer is a special layer for displaying screen buffers. Buffers that can be passed are all of a packed format which means that one pointer is enough to describe the buffer. The interface provides functionality for querying the supported pixel format classes and types, and passing CLUTs and buffers.

In the function specifications below, we use the name layer to denote the instance of the role GfxLayer providing the interface.

Interface ID
```
uuid( 794B33D2-778F-45eb-907F-D664DE2D1508 )
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.7.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIVmixGfxLayerNtf                *pINotify,
    [in] UInt32                            cookie,
    [in] uhVmix_GfxLayerNotifications_t    notifs );
```

Qualifiers
— subscribe-function

8.2.15.5.7.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIVmixGfxLayerNtf                *pINotify,
    [in] UInt32                            cookie,
    [in] uhVmix_GfxLayerNotifications_t    notifs );
```

Qualifiers
— unsubscribe-function

8.2.15.5.7.3 GetSuppFormatClasses

Signature
```
uhErrorCode_t GetSuppFormatClasses (
[out] puhPixFmtClsSet_t     pFormatClasses );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the supported format classes for this graphics layer. The returned integer value represents the OR-ed supported format classes.

Parameters

Table 1286

Name	Description
pFormatClasses	Output parameter returning the supported pixel format classes.

Return Values
Standard.

Precondition
True

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pFormatClasses == layer.suppFormatClasses
```

Remarks

— At least one of the format classes will be supported.

8.2.15.5.7.4 GetSuppFormatTypes

Signature
```
uhErrorCode_t GetSuppFormatTypes (
    [in] uhPixFmtCls_t             formatClass,
    [out] puhPixFmtTypeSet_t       pFormatTypes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the supported format types of the specified pixel format class for this graphics layer. The returned integer value represents the OR-ed supported format types.

Parameters

Table 1287

Name	Description
`formatClass`	The pixel format class for which the supported pixel format types are requested.
`pFormatTypes`	Output parameter returning the supported pixel format types of pixel format class `formatClass`.

Return Values
Standard.

Precondition
```
formatClass in layer.suppFormatClasses
```

Action
None.

Postcondition
```
*pFormatTypes == layer.suppFormatTypes[formatClass]
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.7.5 SetClut

Signature

```
uhErrorCode_t SetClut (
    [in] UInt32                                nrEntries,
    [in, size_is(nrEntries)] uhColorAlpha_t *pClut );
```

Qualifiers

— synchronous

— single-threaded

Description

Sets the Color LookUp Table for this graphics layer. It is the responsibility of the client that the content and size of the table matches the pixel format that will be used. The video mixer will only look at the bits used by the format, e.g. when the buffer format is of type uh`PixClutFmt_RgbClut4bpp` then the CLUT should have 16 entries that have irrelevant alpha values and 8-bits R, 8-bits G, and 8-bits B values.

Parameters

Table 1288

Name	Description
`nrEntries`	The number of elements in array `pClut`.
`pClut`	Pointer to the first element of an array of color lookup table entries.

Return Values

Standard.

Precondition

`True`

Action

— copy `pClut[nrEntries]`

— modify `layer.nextClut`

Postcondition

— `layer.nextClut` == pointer to copy of `pClut[nrEntries]`

Remarks

`SetClut` should be called before `SetBuffer` is called (passing the buffer that uses the CLUT). Until `SetBuffer` is called the platform keeps using the old CLUT `layer.activeClut`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.7.6 GetGfxBufAlignment

Signature
```
uhErrorCode_t GetGfxBufAlignment ( [out] pUInt32  pAlignment );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the byte alignment required of graphics buffer lines that are passed to the layer. The byte alignment poses restrictions on the stride the client specifies for graphics buffers (see `SetBuffer`).

Parameters

Table 1289

Name	Description
`pAlignment`	The byte alignment of a line in a graphics buffer.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
```
*pAlignment == layer.gfxBufAlignment
```

8.2.15.5.7.7 SetBuffer

Signature
```
uhErrorCode_t SetBuffer (
    [in] UInt32                          width,
    [in] UInt32                          height,
    [in] UInt32                          stride,
    [in] uhPixFmtCls_t                   formatClass,
    [in] uhPixFmtType_t                  formatType,
    [in] UInt32                          bufferSize,
    [in, size_is(bufferSize)] pUInt8     pBuffer,
    [in] UInt32                          bufferCookie );
```

Qualifiers
— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Specifies the input image of this graphics layer. The contents of the buffer that is passed will be displayed on the next output composition, unless SetBuffer is called again before that happens. Each time a buffer is passed to the layer the client can get notified of the release of the buffer that is currently in use by the layer (`OnBufferFree`, see subclause 8.2.15.5.8.2). After release of the buffer the client can safely write in the buffer again without producing artifacts on the screen. Setting an empty buffer (`pBuffer` == Null) will release the previous buffer, but what is displayed from the empty buffer is implementation defined.

Parameters

Table 1290

Name	Description
`width`	The number of pixels per line of the image in the buffer.
`height`	The number of lines of the image in the buffer.
`stride`	The stride (in bytes) of the provided buffer. See Figure 157.
`formatClass`	The pixel format class of the image in the buffer.
`formatType`	The pixel format type of the image in the buffer.
`bufferSize`	The size of the buffer in bytes.
`pBuffer`	Pointer to the first element of the buffer where the image is stored (or 0).
`bufferCookie`	An identifier provided by the client that is returned when the client is notified of the disassociation of this buffer and the layer (see function `uhIVmixGfxLayerNtf::OnBufferFree`).

Return Values
Standard.

Precondition
— `if (pBuffer != Null)`
  — `stride >= width * pixel depth of formatType`
  — `stride mod gfxBufAlignment == 0`
  — `bufferSize == stride * height`
  — `formatType` in `layer.suppFormatTypes[formatClass]`
  — `formatClass` in `layer.suppFormatClasses`
  — if `(formatClass == uhPixFmtCls_Clut)`
    — a call to `SetClut` must have been done with pixel formats in conformity with `formatClass` and `formatType`.

Action
— modify `layer.bufferClut, layer.bufferColorKey, layer.nextBuffer`
— Request layer to install buffer (on next field).
— if (`layer.startSettingsOnNextInput == True`)
  — Modify `layer.targetSettings, layer.startSettingsOnNextInput`
  — Request layer to start setting the windows (on next field).

Postcondition
— `layer.nextBuffer == pBuffer`
— `layer.bufferClut == layer.nextClut`
— `layer.bufferColorKey == layer.nextColorKey`
— if `(layer.startSettingsOnNextInput' == True)`
  — `layer.targetSettings == layer.delayedTargetSettings`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— `layer.startSettingsOnNextInput == False`

Remarks

— The first call to `SetBuffer` after a call to `ActivateNewWindowSettings` (subclause 8.2.15.5.3.26) with `syncWithNextInput == True` will trigger the (smooth) transition from `layer.activeSettings` to `layer.targetSettings`.

— If a buffer was already associated with this layer (possibly the same buffer that is passed now), then that buffer will be notified as free as soon as it is not used anymore (see function `uhIVmixGfxLayerNtf::OnBufferFree`).

— A CLUT fade can be implemented by subsequent calls of `SetClut` and `SetBuffer`.

— With `SetBuffer` and the notification function `OnBufferFree` one can implement a double buffering scheme as is depicted in Figure 158.

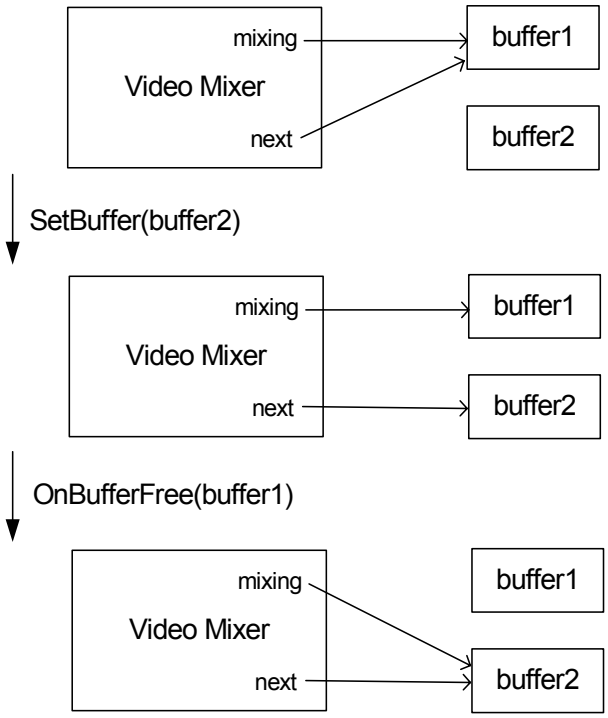


Figure 158 — A double buffering scheme.

8.2.15.5.8 uhIVmixGfxLayerNtf

Qualifiers

— callback

Description

This interface should be provided by each client that wants to subscribe to notifications of events that occur in a

GfxLayer. The Subscribe function of `uhIVmixGfxLayer` can be used to subscribe to these events.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Interface ID
`uuid( 90CBCFD1-3D49-408b-A8BC-B0A00EE2A829 )`

8.2.15.5.8.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32                          cookie,
    [in] uhVmix_GfxLayerNotifications_t  notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.2.15.5.8.2 OnBufferFree

Signature
```
[async] Void OnBufferFree (
    [in] UInt32       cookie,
    [in] UInt32       bufferCookie );
```

Qualifiers
None.

Description
This function is called by a `GfxLayer` as soon as the buffer identified by `bufferCookie` is no longer associated with the layer. The client can now write in this buffer without producing (unwanted) effects on the screen.

Parameters

Table 1291

Name	Description
`cookie`	The cookie identifying the subscription that enabled this notification.
`bufferCookie`	The value passed by the client in a call of function `SetBuffer` (see `uhIVmixGfxLayer::SetBuffer`).

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.9 uhIVmixColorKey

Qualifiers
None.

Description
A layer has a fixed number of color keys that are numbered starting at 0. The `uhIVmixColorKey` interface associated with the layer role provides access to these color keys. The color key number is used as an identification in the functions of this interface.

The interface provides functionality for querying the supported number of color keys, the supported color key types, and setting, getting, enabling, and disabling a color key.

In the function specifications below, we use the name layer to denote the instance of the role Layer providing the interface.

Interface ID
`uuid( C61ACC8E-33B5-4fdb-A765-9DD6B2F674DB )`

8.2.15.5.9.1 GetSuppNrKeys

Signature
```
uhErrorCode_t GetSuppNrKeys ( [out] pUInt32 pNrKeys );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the number of color keys supported by the layer.

Parameters

Table 1292

Name	Description
`pNrKeys`	Output parameter returning the supported number of color keys.

Return Values
Standard.

Precondition
`True`

Action
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
```
*pNrKeys == layer.nrColorKeys
```

8.2.15.5.9.2 GetSuppKeyTypes

Signature
```
uhErrorCode_t GetSuppKeyTypes (
    [out] puhVmix_ColorKeyTypeSet_t      pSuppKeyTypes );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the color key types supported by the layer. There are two types of color keying: *source* color keying and *destination* color keying; see subclause 8.2.15.1.5.

Parameters

Table 1293

Name	Description
`pSuppKeyTypes`	Output parameter returning the supported color key types.

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
*pSuppKeyTypes == layer.suppColorKeyTypes
```

8.2.15.5.9.3 SetKey

Signature
```
uhErrorCode_t SetKey (
    [in] UInt32                keyNr,
    [in] uhVmix_ColorKeyType_t keyType,
    [in] uhColor_t             lowerColor,
    [in] uhColor_t             upperColor );
```

Qualifiers
— synchronous

— single-threaded

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Sets one of the color keys of a layer. The client needs to specify the type of color key (source or destination) and the color key range itself (which should be specified in YUV for video layers and RGB/YUV for graphics layers, depending on the format of the buffer to which the color key is applied). When set, the color key is not necessarily active. Color keys have to be enabled explicitly with `EnableKey` (subclause 8.2.15.5.9.5).

Parameters

Table 1294

Name	Description
`keyNr`	The number of the color key to be set.
`keyType`	The type of color key (source or destination).
`lowerColor`	The lower limit of the color key range.
`upperColor`	The upper limit of the color key range.

Return Values
Standard.

Precondition
— `0 <= keyNr < layer.nrColorKeys`
— `keyType` in `layer.suppColorKeyTypes`

Action
— Modify `layer.nextColorKey[keyNr]`

Postcondition
— `layer.nextColorKey[keyNr].keyType == keyType`
— `layer.nextColorKey[keyNr].lowerColor == lowerColor`
— `layer.nextColorKey[keyNr].upperColor == upperColor`

Remarks
— Comparison of pixels with the color key upper and lower limits is done component wise. For example, on a graphics layer displaying an RGB buffer the following will have to hold for pixel in order to generate a color key hit: R(*pixel*) >= R(`lowerColor`), and G(*pixel*) >= G(`lowerColor`), and B(*pixel*) >= B(`lowerColor`), and R(*pixel*) <= R(`upperColor`), and G(*pixel*) <= G(`upperColor`), and B(*pixel*) <= B(`upperColor`). Note that the color space used by the color key depends on the buffer contents, so that, if the client passes a YUV buffer without changing the color key, the upper and lower limits will then be interpreted as YUV.

— The client may set `upperColor` "smaller" than `lowerColor`. This is an empty range; a color key hit will never be generated on this range.

— On a video layer the color key update is done on a new field; on a graphics layer the color key update is done when a (new) buffer is passed with `SetBuffer` (see subclause 8.2.15.5.7.7).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.15.5.9.4 GetKey

Signature

```
uhErrorCode_t GetKey (
    [in] UInt32                  keyNr,
    [out] puhVmix_ColorKeyType_t pKeyType,
    [out] puhColor_t             pLowerColor,
    [out] puhColor_t             pUpperColor );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns one of the color keys of a layer as set by the client.

Parameters

Table 1295

Name	Description
`keyNr`	The number of the color key to returned.
`pKeyType`	Output parameter returning the type of color key (source or destination).
`pLowerColor`	Output parameter returning the lower limit of the color key range.
`pUpperColor`	Output parameter returning the upper limit of the color key range.

Return Values

Standard.

Precondition

```
0 <= keyNr < layer.nrColorKeys
```

Action

None.

Postcondition

— `*pKeyType == layer.nextColorKey[keyNr].keyType`
— `*pLowerColor == layer.nextColorKey[keyNr].lowerColor`
— `*pUpperColor == layer.nextColorKey[keyNr].upperColor`

8.2.15.5.9.5 EnableKey

Signature

```
uhErrorCode_t EnableKey ( [in] UInt32 keyNr, [in] Bool enable );
```

Qualifiers

— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— single-threaded

Description
Enables/disables one of the color keys of a layer.

Parameters

Table 1296

Name	Description
`keyNr`	The number of the color key to be enabled/disabled.
`enable`	Determines if the color key has to be enabled (`True`) or disabled (`False`).

Return Values
Standard.

Precondition
```
0 <= keyNr < layer.nrColorKeys
```

Action
— Modify `layer.nextColorKey[keyNr].enabled`

Postcondition
```
layer.nextColorKey[keyNr].enabled == enable
```

Remarks
On a video layer the actual color key update is done on a new field; on a graphics layer the actual color key update is done when a (new) buffer is passed with `SetBuffer` (see subclause 8.2.15.5.7.7).

8.2.15.5.9.6 GetKeyEnabled

Signature
```
uhErrorCode_t GetKeyEnabled ( [in] UInt32 keyNr, [out] pBool pEnabled );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the specified color key is enabled or not.

Parameters

Table 1297

Name	Description
`keyNr`	The number of the color key.
`pEnabled`	Output parameter returning whether the color key is enabled (`True`) or disabled (`False`).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`0 <= keyNr < layer.nrColorKeys`

Action
None.

Postcondition
`*pEnabled == layer.nextColorKey[keyNr].enabled`

8.2.15.5.10 uhIVmixBorder

Qualifiers
None.

Description
A layer optionally supports borders. The `uhIVmixBorder` interface associated with the layer role provides means to manipulate these borders.

The interface provides functionality for defining and querying the color of the borders and the thickness of each separate border (top, bottom, left, and right). Borders can be shown or hidden.

In the function specifications below, we use the name layer to denote the instance of the role Layer providing the interface.

Interface ID
`uuid( 649C06C2-58D6-444f-8181-C909AE1380CD )`

8.2.15.5.10.1 GetHidden

Signature
```
uhErrorCode_t GetHidden ( [out] pBool pHidden );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns whether the borders are shown or hidden.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1298

Name	Description
`pHidden`	Output parameter returning whether the borders are shown (`False`) or hidden (`True`).

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*pHidden == layer.borders.hidden`

8.2.15.5.10.2 Hide

Signature
```
uhErrorCode_t Hide ( [in] Bool  hide );
```

Qualifiers
— synchronous

— single-threaded

Description
Hides or shows the borders of a layer.

Parameters

Table 1299

Name	Description
`hide`	Borders of the layer are shown (`False`) or hidden (`True`).

Return Values
Standard.

Precondition
`True`

Action
— Modify `layer.borders.hidden`
— Request the layer to hide/unhide its borders (on the next field).

©ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
```
layer.borders.hidden == hide
```

8.2.15.5.10.3 Get

Signature
```
uhErrorCode_t Get (
    [out] pUInt32    pTop,
    [out] pUInt32    pBottom,
    [out] pUInt32    pLeft,
    [out] pUInt32    pRight,
    [out] puhColor_t pColor );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the (RGB) color of the borders and the thickness (in pixels/lines) of each separate border.

Parameters

Table 1300

Name	Description
`pTop`	Output parameter returning the thickness (in lines) of the top border.
`pBottom`	Output parameter returning the thickness (in lines) of the bottom border.
`pLeft`	Output parameter returning the thickness (in pixels) of the left border.
`pRight`	Output parameter returning the thickness (in pixels) of the right border.
`pColor`	Output parameter returning the color of the layer borders.

Return Values
Standard.

Precondition
```
True
```

Action
None.

Postcondition
```
— *pTop == layer.borders.top
— *pBottom == layer.borders.bottom
— *pLeft == layer.borders.left
— *pRight == layer.borders.right
— *pColor == layer.borders.color
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks
— Although the RGB color value set with Set is converted to the pixel format used on this layer, the original value that was set is returned when the border color is queried.

8.2.15.5.10.4 Set

Signature
```
uhErrorCode_t Set (
    [in] UInt32    top,
    [in] UInt32    bottom,
    [in] UInt32    left,
    [in] UInt32    right,
    [in] uhColor_t color );
```

Qualifiers
— synchronous

— single-threaded

Description
Sets the (RGB) color of the borders and the thickness (in pixels/lines) of each separate border.

Parameters

Table 1301

Name	Description
`top`	Thickness (in lines) of the top border.
`bottom`	Thickness (in lines) of the bottom border.
`left`	Thickness (in pixels) of the left border.
`right`	Thickness (in pixels) of the right border.
`color`	The layer border color to be set.

Return Values
Standard.

Precondition
`True`

Action
— Modify `layer.borders.top`, `layer.borders.bottom`, `layer.borders.left`, `layer.borders.right`, `layer.borders.color`
— Request the layer to change the color and thickness of its borders (on the next field).

Postcondition
— `layer.borders.top == top`
— `layer.borders.bottom == bottom`
— `layer.borders.left == left`
— `layer.borders.right == right`
— `layer.borders.color == color`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Remarks

— The thicknesses of the borders may be chosen such that opposite borders overlap, or that a border is bigger than the size of the layer window. In such a case, the complete layer (if visible) will show the border color.

— There is a default layer border color configured at startup time.

— color will be converted to the pixel format currently used for this layer.

8.3 Personal video recording

8.3.1 UhlDataInject

8.3.1.1 Concepts

8.3.1.1.1 Introduction

The function of a Data Injector is to facilitate the injection of any kind of data from its client into the platform.

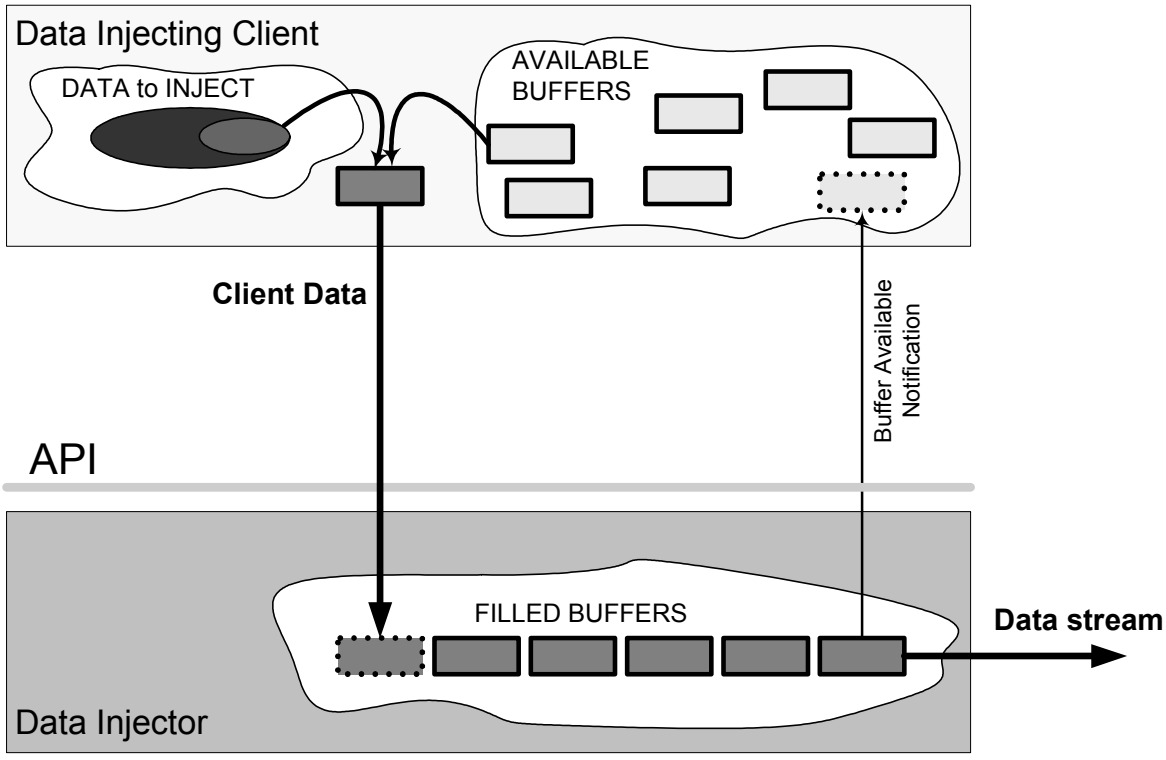


Figure 159 — Flow of data through the Data Injecting Logical Component

In Figure 159, a schematic representation of the data flow through the Data Injecting Logical Component is depicted.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The client of the Logical Component injects the data into the platform via buffers: the injection is performed by the client by adding the data to available (free) buffers and then by triggering the appropriate function on the interface provided by the platform.

To prevent extra and unnecessary copying, the data to inject can be directly added to available buffers when, for instance, the client reads the data from a DVD Disk or an MP3 file.

The Data Injecting Client is handling the available (free) buffers, while the filled buffers are handled by the Data Injector: this is indicated in Figure 159 by placing the available buffers inside the Data Injecting Client and the filled buffers inside the Data Injector, respectively.

Note that Figure 159 does not show the component that is actually managing (i.e. allocating/de-allocating) the buffers. The Data Injector supports both buffers managed by the client and buffers managed by the platform.

The "streaming" output of the Data Injecting Logical Component contains exactly the data that has been injected by the client. The data is streamed out in the same order as it has been injected.

The buffers, to which data has been added for injection, become free for further addition of new data after the Data Injecting Logical Component has streamed out the contained data. This is indicated to the client by means of a “buffer available” notification.

8.3.1.1.2 Data Injection

The Data Injecting Logical Component is a generic component: it can handle the injection of any kind of data since the Data Injector does not perform any parsing or interpretation of the data that is injected by its client.

This Logical Component can handle the injection of data in the PVR context; it can also support the injection of data that was read by the client from a DVD disk, or a MP3 file.

A first consequence is that the Data Injector does not provide any functionality to its client(s) to add meta-data to the injected data. If needed, the client should provide this meta-data to a downstream component that is interpreting the data, or it should add the meta-data itself to the data before injection.

Another consequence is that the Data Injector does not provide any explicit means to its client(s) to control directly when the data is streamed out. There is no control provided on the interfaces specified in this document for this, and if a client adds time-stamps to the injected data (or if the data already contains time-stamps), these are not interpreted by the Data Injector, but just streamed out with the rest of the data (a downstream component may be able though to interpret the time-stamps).

Finally, any error in the injected data cannot be detected by the Data Injector since the latter does not interpret any of the data that is injected (a downstream component may be able though to detect errors in the injected data).

8.3.1.1.2.1 Buffers

Injection of the required data into the platform is performed via buffers.

These buffers are provided by the platform and/or by the client itself.

In principle, the Data Injecting Logical Component has a sufficient amount of buffers available that can be used for the injection of data into the platform: an optional, but recommended, interface specified in this document provides a means to get these buffers. Whether these buffers are allocated statically or dynamically is up to the platform instance. The number, the alignment, the location in memory, and size of these buffers as well.

In addition, if the platform allows it, the client may manage himself some of the buffers (i.e. the allocation/de-allocation). Sometimes, the client should even manage all the buffers himself: this is the case when the platform does not contribute to the buffer management at all.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the buffers are managed only by the platform, the Data Injecting Client must add the (filled) buffers in the same order as he got them from the platform; this allows the use of ring buffers in the platform. When some of the buffers are managed by the Data Injecting Client itself then this constraint is not applicable. The client may define its own buffer management and may use the most appropriate buffer(s) for the data to be currently injected.

The lifetime of the buffers that are managed by the platform itself is the same as the lifetime of the associated Logical Component instance.

During that lifetime, buffers are always available for data injection into the platform, apart - of course - when they are already in use to inject data and they have not yet been notified as freed by means of a notification by the Data Injector.

The buffers that are managed by the client do not necessarily have identical sizes, while the buffers managed by the platform have identical sizes. The client should use the full buffer capacity, apart when the data to inject is the last data of the currently injected stream of data. The client can also inject data via several buffers at the same time.

The buffers, to which data has been added for injection, become free for further addition of new data as soon as possible (unless otherwise specified in the Platform Instance document) after the Data Injecting Logical Component has streamed out the contained data. This is indicated to the client by means of a notification. It is possible that a single notification indicates that several buffers are from this time on available for further addition of new data.

The buffers are becoming free for further addition of new data, and are notified, in the same order as they were added by the Data Injecting Client.

A means is provided to the client to indicate that the injected data is the last data of the currently injected stream of data: this can be of interest for the processing that can possibly happen further downstream this Logical Component.

8.3.1.1.2.2 Output Stream

The Data Injector streams the data out in the same order as it has been injected.

Depending on the use case, the Data Injector may stream out the injected data right away - for instance, when the injected data is an MP3 file and the component further downstream is an audio decoder - or as soon as the component further downstream asks for new data.

The pace with which the data is streamed out is not necessarily directly determined by the pace at which the client is injecting data via the buffers. It might also depend on the behavior of the component(s) further downstream. There is also no control provided to the client to regulate the pace at which data is streamed out.

The client should however take care itself that there is sufficient data injected so that a possible processing further downstream can be done correctly: the Data Injecting Logical Component is not able to detect whether the pace of injected data is too low for a correct processing further downstream.

Consequently, the Data Injecting Logical Component cannot detect and therefore cannot indicate to the client that it is almost facing a data underflow situation.

8.3.1.2 Types & Constants

8.3.1.2.1 Public Types & Constants

8.3.1.2.1.1 Error Codes

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.1.2.1.2 uhDataInject_Ntf_t

Signature

```
typedef enum _uhDataInject_Ntf_t {
    uhDataInject_SubscriptionChanged = 0x00000001,
    uhDataInject_BuffersAvailable    = 0x00000002
} uhDataInject_Ntf_t, *puhDataInject_Ntf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIDataInjectNtf`.

Values

Table 1302

Name	Description
`uhDataInject_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhDataInject_BuffersAvailable`	Represents `OnBuffersAvailable`.

8.3.1.2.1.3 uhDataInject_NtfSet_t

Signature

```
typedef Int32 uhDataInject_NtfSet_t, *puhDataInject_NtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhDataInject_`Ntf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints

None.

8.3.1.2.1.4 uhDataInject_AllNtfs

Signature

```
const uhDataInject_NtfSet_t uhDataInject_AllNtfs = 0x00000003;
```

Qualifiers

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set of all values of type uhDataInject_Ntf_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.3.1.2.2 Model Types & Constants

8.3.1.2.2.1 fifoList

Signature
```
typedef struct {
    UInt32 maxNrBuffers;
    UInt32 nrBuffers;
    UInt32 buffers[maxNrBuffers];
    UInt32 buffersId[maxNrBuffers];
    UInt32 buffersLength[maxNrBuffers];
    UInt32 posStreamingOut;
} fifoList;
```

Qualifiers
None.

Description
Values of this type represent a sorted list of pointers to the start address of buffers: these buffers have been provided by a Data Injecting Client and are (or just were) containing data; this data is meant to be streamed out by the Data Injector.

The first element(s) of this list are always pointing to the buffer(s) for which the data has (just) been streamed out and that should still be released and thus notified (individually or in groups), since they are now available for injection of new data. Then come in this list the pointers to the buffers that contain data that should (still) be streamed out by the Data Injector.

The position of the "first" buffer for which the data should (still) be streamed out by the Data Injector is indicated by the posStreamingOut variable.

Once the data of this “first” buffer has been streamed out by the Data Injector, this buffer can be released and notified as available for injection of new data to the Data Injecting Client (or notified later, e.g. if several buffers are notified together). Consequently, the previously "second" buffer in the array for which the data should be streamed out becomes the "first" buffer in the array for which the data should be streamed out (the value of posStreamingOut is increased by one), and so on, once the contained data of this “new first” buffer is also streamed out.

When a number of (available) buffers are notified, they are removed from the sorted list buffers (their identifying cookie in buffersId and their size in buffersLength as well), and the value of posStreamingOut (as well as the value of the nrBuffers variable) is decreased by this number.

Fields

Table 1303

Name	Description
maxNrBuffers	The maximum number of pointers that the array buffers can contain.
nrBuffers	The number of pointers that the array buffers actually contains.
buffers	Array containing pointers to the start address of buffers.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
buffersId	Array containing cookies that identify each of the buffers in buffers. buffersId[i] identifies the buffer pointed by buffers[i]. The cookies are meant to simplify the buffer management task of the Data Injecting Client.
buffersLength	Array containing with how bytes of data each buffer pointed by buffers is filled. buffersLength[i] contains with how many bytes of data the buffer pointed by buffers[i] is filled.
posStreamingOut	The index from which the buffers in buffers should (still) be streamed out by the Data Injector. Buffers with indexes in {0..posStreamingOut-1} contained data that has (just) been streamed out and they should be released and notified as available for injection of new data to the Data Injecting Client.

Constraints

- posStreamingOut ≤ nrBuffers ≤ maxNrBuffers
- for i in [nrBuffers..maxNrBuffers-1]
  - buffers[i] == 0
  - buffersId[i] == 0
  - buffersLength[i] == 0

8.3.1.3 Logical Component

8.3.1.3.1 Interface-Role Model

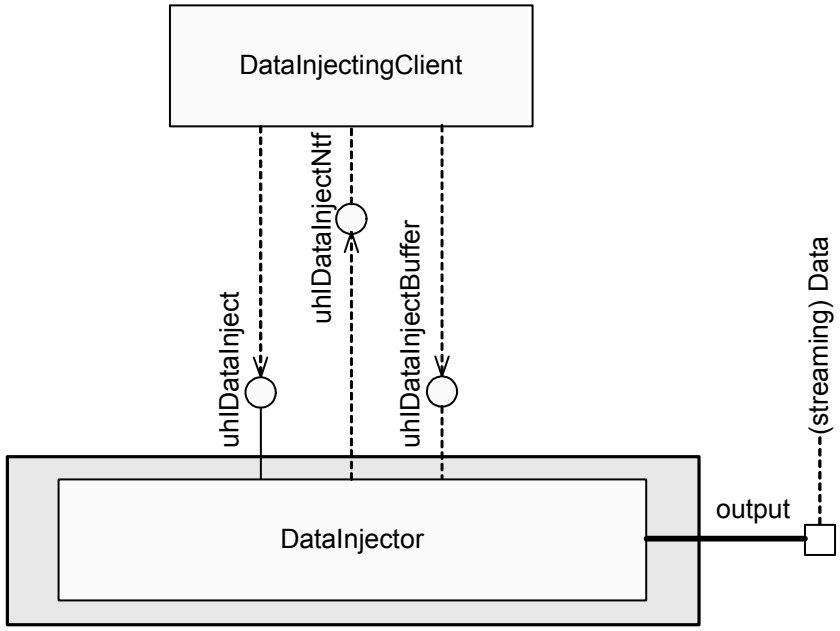


Figure 160 — Interface Role Model

The grey box in Figure 160 denotes a Data Injecting Logical Component containing a single Data Injector role.

This role provides one mandatory (solid line) control interface (uhIDataInject) and one optional (dotted line) control interface (uhIDataInjectBuffer).

A client can optionally provide a call-back (dotted lines) interface uhIDataInjectNtf. This allows the client to receive notifications of events that occur in the Data Injector.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.1.3.2 Diversity

8.3.1.3.2.1 Provided Interfaces

Table 1304

Role	Interface	Presence
DataInjector	uhIDataInject	mandatory
DataInjectingClient	uhIDataInjectNtf	optional
DataInjector	uhIDataInjectBuffer	optional

8.3.1.3.2.2 Configurable Items

Table 1305

Role	Attribute
DataInjector	maxNrBuffers
	clientBuffersAllowed
	clientBuffersMinSize
	clientBuffersMaxSize
	clientBuffersAlignment
	nrPlatfBuffers
	platfBuffersSize
	platfBuffersSharable
	platfBuffers

8.3.1.3.2.3 Constraints

```
— maxNrBuffers > 0
— clientBuffersMinSize ≤ clientBuffersMaxSize
— if (!clientBuffersAllowed)
   — clientBuffersMinSize == 0
   — clientBuffersMaxSize == 0
   — clientBuffersAlignment == 0
— else
   — clientBuffersMinSize > 0
   — clientBuffersMaxSize > 0
   — clientBuffersAlignment > 0
— if nrPlatfBuffers > 0
   — platfBuffersSize > 0
— else
   — platfBuffersSharable == False
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.1.3.3 Instantiation

8.3.1.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1306

Type	Object	Multiplicity
DataInjector	dataInj	1

8.3.1.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

- dataInj.buffersToStreamOut.maxNrBuffers == maxNrBuffers
- dataInj.buffersToStreamOut.nrBuffers == 0
- dataInj.buffersToStreamOut.posStreamingOut == 0
- for i in [0..maxNrBuffers-1]
  - dataInj.buffersToStreamOut.buffers == 0
  - dataInj.buffersToStreamOut.buffersId == 0
  - dataInj.buffersToStreamOut.buffersLength == 0

8.3.1.3.4 Execution Constraints

The interfaces provided by the roles of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.3.1.4 Roles

8.3.1.4.1 DataInjector

Signature

```
role DataInjector  {
    const UInt32      maxNrBuffers;
    const Bool        clientBuffersAllowed;
    const UInt32      clientBuffersMinSize;
    const UInt32      clientBuffersMaxSize;
    const UInt32      clientBuffersAlignment;
    const UInt32      nrPlatfBuffers;
    const UInt32      platfBuffersSize;
    const UInt32      platfBuffersSharable;
    const UInt32      platfBuffers[nrPlatfBuffers];
    fifoList          buffersToStreamOut;
    const Bool        orderImportant;
}
```

Qualifiers

- root

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The Data Injector role provides means to inject any kind of data into the platform to its client(s) and streams out without any interpretation the injected data to downstream components.

The Data Injector role may manage (i.e. allocate/de-allocate) buffers and provide them to client(s) so that they can be used to inject data into the platform.

Independent Attributes

Table 1307

Name	Description
`maxNrBuffers`	The maximum of filled buffers that can be added by the Data Injecting Client into the platform, and that can be administrated simultaneously by the Data Injector.
`clientBuffersAllowed`	Indication on whether the Data Injecting Client is allowed to manage buffers itself and to use these buffers to inject data into the platform. It is `True` when the Data Injecting Client is allowed to manage buffers itself; it is `False` when the Data Injecting Client is not allowed to manage buffers itself.
`clientBuffersMinSize`	The minimum size that the buffers managed by the Data Injecting Client should have.
`clientBuffersMaxSize`	The maximum size that the buffers managed by the Data Injecting Client should have.
`clientBuffersAlignment`	The alignment that the buffers managed by the Data Injecting Client should have.
`nrPlatfBuffers`	The number of buffers that are managed by the Data Injector and that can be used by the Data Injecting Client to inject data.
`platfBuffersSize`	The size of the buffers managed by the Data Injector.
`platfBuffersSharable`	Indication on whether (platform) buffers from other Data Injector(s)/Data Extractor(s) can be used with this Data Injector. It is `True` if (platform) buffers from other Data Injector(s)/Data Extractor(s) can be used with this Data Injector. It is `False` if (platform) buffers from other Data Injector(s)/Data Extractor(s) can't be used with this Data Injector.
`platfBuffers`	Array containing the start address of the buffers that are managed by the Data Injector and that can be used by the Data Injecting Client to inject data.
`buffersToStreamOut`	Sorted list containing pointers to the start address of buffers that have been injected by the Data Injecting Client and for which the contained data still needs to be streamed out by the Data Injector to downstream components.

Dependent Attributes

Table 1308

Name	Description
`orderImportant`	Indication on whether the order in which the buffers are added is important. It is equal to `{!dataInj.clientBuffersAllowed && !dataInj.platfBuffersSharable}`. If it is `True` then `the Data Injecting Client must add the (platform) buffers in the same order as he got them from the platform.` `If it is False then the order in which the buffers that are managed by the platform are injected, is not relevant.`

Invariants
`buffersToStreamOut.maxNrBuffers == maxNrBuffers`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Instantiation
A Data Injector is always instantiated as part of a Data Injecting LC; see subclause 8.3.1.3.3.

Streaming Behavior
The Data Injector has no streaming input: it gets data from the Data Injecting Client via buffers.

The "streaming" output of the Data Injecting Logical Component contains exactly the data that has been injected by the client.

Depending on the use case, the Data Injector may stream out the injected data right away - for instance, when the injected data is an MP3 file and the component further downstream is an audio decoder - or as soon as the component further downstream asks for new data.

The Data Injector streams the data out in the same order as it has been injected.

A means is provided to the client to indicate that the injected data is the last data of the currently injected stream of data: this can be of interest for the processing that can possibly happen further downstream.

Active Behavior

Table 1309

Event	Action
The data contained in one or several buffers has been streamed out, the buffer(s) became free for further addition of new data to inject, the Data Injector is ready to notify the available buffer(s).	Let `n` be the number of buffers for which data has been streamed out and that should be released and notified as available for further addition of new data to inject. Let `buffersId` be the array of cookies identifying each of the available buffers to be (currently) notified — Notify `OnBuffersAvailable(n,buffersId)` — The `n` first elements from the sorted list `dataInj.buffersToStreamOut.buffers` are removed (their identifying cookie in `dataInj.buffersToStreamOut.buffersId` and their size in `dataInj.buffersToStreamOut.buffersLength` as well), and the value of `dataInj.buffersToStreamOut.posStreamingOut` (as well as the value of the `dataInj.buffersToStreamOut.nrBuffers` variable) is decreased by `n`.

Execution Constraints
None.

8.3.1.4.2 DataInjectingClient

Signature
```
role DataInjectingClient {}
```

Qualifiers
— actor

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
A Data Injecting Client represents a user of the following interfaces: `uhIDataInject` and `uhIDataInjectBuffer`.

Execution Constraints
None.

8.3.1.5 Interfaces

8.3.1.5.1 uhIDataInject

Qualifiers
None.

Description
This interface of a Data Injector provides facilities to add data into the platform via buffers. It also provides facilities to retrieve the capabilities and the constraints of the Data Injector with respect to buffers managed by the Data Injecting Client itself. Finally, it provides a `Subscribe` and an `Unsubscribe` function for notifications of events related to the management of buffers.

Interface ID
`uuid( F5A68D60-AE31-4BEA-901A-FD85A759810D )`

8.3.1.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIDataInjectNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhDataInject_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

8.3.1.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIDataInjectNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhDataInject_NtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.1.5.1.3 GetMaxNrBuffers

Signature

```
uhErrorCode_t GetMaxNrBuffers (
    [out] pUInt32 pMaxNrBuffers );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the maximum number of filled buffers that can be added by the Data Injecting Client into the platform and for which the data is meant to be streamed out by the Data Injector to downstream components.

Parameters

Table 1310

Name	Description
`pMaxNrBuffers`	Output parameter returning the maximum of filled buffers that can be added by the Data Injecting Client into the platform, and that can be administrated simultaneously by the Data Injector.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

`*pMaxNrBuffers == dataInj.maxNrBuffers`

8.3.1.5.1.4 GetActualNrBuffers

Signature

```
uhErrorCode_t GetActualNrBuffers (
    [out] pUInt32 pActualNrBuffers );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the actual number of filled buffers that have been added by the Data Injecting Client into the platform and for which the data is still meant to be streamed out by the Data Injector to downstream components.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1311

Name	Description
pActualNrBuffers	Output parameter returning the actual number of filled buffers that have been added by the Data Injecting Client into the platform and for which the data is still meant to be streamed out by the Data Injector to downstream components.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pActualNrBuffers == dataInj.buffersToStreamOut.nrBuffers

8.3.1.5.1.5 AddData

Signature

```
uhErrorCode_t AddData (
    [in] UInt32 nrBuffers,
    [in, size_is(nrBuffers)] const UInt32* pBuffers,
    [in, size_is(nrBuffers)] const UInt32* pBuffersId,
    [in, size_is(nrBuffers)] const UInt32* pBuffersLength,
    [in] Bool endOfStream );
```

Qualifiers
— synchronous

— single-threaded

Description
Adds a specified number of filled buffers. The contained data is meant to be streamed out by the Data Injector to downstream components. The Data Injecting Client can indicate that the injected data is the last data of the currently injected stream of data.

A Data Injecting Client should not modify any buffer that has been added, until it has received the buffer back through an OnActEnabledChanged notification.

If the Data Injecting Client is allowed to manage buffers,

— these can be used and therefore added successively with or simultaneously with (i.e. in the same AddData function call) the buffers that are managed by the Data Injector;

— the sizes of the buffers that are added successively (respectively, simultaneously) are not necessarily identical, a Data Injecting Client may add a buffer with size 2kb after (respectively, along with) a buffer with size 4kb for instance;

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— the order in which the platform buffers are added is not relevant anymore.

Parameters

Table 1312

Name	Description
nrBuffers	The number of filled buffers containing data that is meant to be injected into the platform.
`pBuffers`	Pointer to the first element of an array of pointers to the start address of buffers containing the data that is meant to be injected in the platform.
`pBuffersId`	Pointer to the first element of an array of cookies: each cookie identifies one of the added buffers, and each cookie is passed to the Data Injecting Client by the Data Injector when notifying an event related to the related buffer. `pBuffersId[i]` identifies the buffer pointed to by `pBuffers[i]`. The cookies are meant to simplify the buffer management task of the Data Injecting Client.
`pBuffersLength`	Pointer to the first element of an array containing with how bytes of data each buffer pointed to by `pBuffers` is filled. `pBuffersLength[i]` contains with how many bytes of data the buffer pointed by `pBuffers[i]` is filled.
`endOfStream`	Indication that the injected data is the last data of the currently injected stream of data. It is `True` if the injected data is the last data of the currently injected stream of data. It is `False` if not.

Return Values
Standard.

Precondition
- `0 < nrBuffers ≤ dataInj.maxNrBuffers -dataInj.buffersToStreamOut.nrBuffers`
- `if (dataInj.orderImportant)`
  - `the Data Injecting Client must add the (platform) buffers in the same order as he got them from the platform`
- `else`
  - `the order in which the buffers that are managed by the platform are injected, is not relevant`
- `for i in {0..nrBuffers-1}`
  - `if pBuffers[i] is pointing to the start address of a buffer managed by the platform`
    - `(pBuffersLength[i] == dataInj.platfBuffersSize) || (endOfStream && (pBuffersLength[i] ≤ dataInj.platfBuffersSize))`
  - `else`
    - `(dataInj.clientBuffersMinSize ≤ pBuffersLength[i] ≤ dataInj.clientBuffersMaxSize) || (endOfStream && (pBuffersLength[i] ≤ dataInj.clientBuffersMaxSize))`
    - `The alignment of the buffers of which pBuffers[i] is pointing to the start address is identical to dataInj.clientBuffersAlignment`
  - `for j in {0..nrBuffers-1}, j!=i`
    - `pBuffers[i] != pBuffers[j]`
  - `for j in {0..dataInj.buffersToStreamOut.nrBuffers-1}`
    - `pBuffers[i] != dataInj.buffersToStreamOut.buffers[j]`

Action
- Modify `dataInj.buffersToStreamOut.nrBuffers`
- Modify `dataInj.buffersToStreamOut.buffers`
- Modify `dataInj.buffersToStreamOut.buffersId`
- Modify `dataInj.buffersToStreamOut.buffersLength`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition

— `dataInj.buffersToStreamOut.nrBuffers == dataInj.buffersToStreamOut.nrBuffers'+nrBuffers`
— `For i in {0..nrBuffers-1}`
  — `dataInj.buffersToStreamOut.buffers [dataInj.buffersToStreamOut.nrBuffers'+i] == pBuffers[i]`
  — `dataInj.buffersToStreamOut.buffersId [dataInj.buffersToStreamOut.nrBuffers'+i] == pBuffersId[i]`
  — `dataInj.buffersToStreamOut.buffersLength [dataInj.buffersToStreamOut.nrBuffers'+i] == pBuffersLength[i]`

Remarks

The `endOfStream` indication can be useful for some (further downstream) decoders that only decode a next frame, if their input buffer is not "full enough". At the end of a stream, the decoder buffer in the implementation may not be full enough to decode the last frame. With the `endOfStream` indication, the decoder can be instructed to decode the last frame as well.

8.3.1.5.1.6 FlushData

Signature

```
uhErrorCode_t FlushData ();
```

Qualifiers

— asynchronous

— single-threaded

Description

Flushes all the filled buffers that have been added by the Data Injecting Client, and that have not yet been notified by the Data Injector as free for further addition of new data to inject.

All the buffers that have been flushed are returned back to the Data Injecting Client through a single `OnActEnabledChanged` notification, in the same order as they were added.

Parameters

None.

Return Values

Standard.

Precondition

`True`

Asynchronous Action

— Modify `dataInj.buffersToStreamOut.nrBuffers`
— Modify `dataInj.buffersToStreamOut.buffers`
— Modify `dataInj.buffersToStreamOut.buffersId`
— Modify `dataInj.buffersToStreamOut.buffersLength`
— Modify `dataInj.buffersToStreamOut.posStreamingOut`
— `Notify OnBuffersAvailable(dataInj.buffersToStreamOut.nrBuffers', dataInj.buffersToStreamOut.buffersId')`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Asynchronous Postcondition

- `dataInj.buffersToStreamOut.nrBuffers == 0`
- `dataInj.buffersToStreamOut.posStreamingOut == 0`
- For `i` in `{0..dataInj.maxNrBuffers-1}`
  - `dataInj.buffersToStreamOut.buffers[i] == 0`
  - `dataInj.buffersToStreamOut.buffersId[i] == 0`
  - `dataInj.buffersToStreamOut.buffersLength[i] == 0`

8.3.1.5.1.7 GetClientBufferInfo

Signature

```
uhErrorCode_t GetClientBufferInfo (
    [out] pBool pClientBuffersAllowed,
    [out] pUInt32 pMinBuffersSize,
    [out] pUInt32 pMaxBuffersSize,
    [out] pUInt32 pBuffersAlignment );
```

Qualifiers

- synchronous
- thread-safe

Description

Returns whether the Data Injecting Client is allowed to manage buffers and to use them for injection of data into the platform. It also returns the minimum and the maximum size the buffers can have and their required alignment, in case the Data Injecting Client is allowed to manage buffers and to use them.

Parameters

Table 1313

Name	Description
`pClientBuffersAllowed`	Output parameter returning whether the Data Injecting Client is allowed to manage buffers itself and to use these buffers to inject data into the platform. It is `True` when the Data Injecting Client is allowed to manage buffers itself; it is `False` when the Data Injecting Client is not allowed to manage buffers itself.
`pMinBuffersSize`	Output parameter returning the minimum size that the buffers managed by the Data Injecting Client should have. It is `0` when the Data Injecting Client is not allowed to manage buffers itself.
`pMaxBuffersSize`	Output parameter returning the maximum size that the buffers managed by the Data Injecting Client should have. It is `0` when the Data Injecting Client is not allowed to manage buffers itself.
`pBuffersAlignment`	Output parameter returning the alignment that the buffers managed by the Data Injecting Client should have. It is `0` when the Data Injecting Client is not allowed to manage buffers itself.

Return Values

Standard.

Precondition

`True`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
— *pClientBuffersAllowed == dataInj.clientBuffersAllowed
— *pMinBuffersSize == dataInj.clientBuffersMinSize
— *pMaxBuffersSize == dataInj.clientBuffersMaxSize
— *pBuffersAlignment == dataInj.clientBuffersAlignment

8.3.1.5.2 uhIDataInjectNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in the Data Injector. The `Subscribe` function of the `uhIDataInject` can be used to subscribe to these events.

Interface ID
`uuid( 2A2FC3F8-EEA2-427B-9B90-C538FC181507 )`

8.3.1.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhDataInject_NtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.3.1.5.2.2 OnBuffersAvailable

Signature
```
[async] Void OnBuffersAvailable (
    [in] UInt32 cookie,
    [in] UInt32 nrBuffers,
    [in, size_is(nrBuffers)] const UInt32* pBuffersId );
```

Qualifiers
None.

Description
This function is called by the Data Injector whenever buffers, which were filled with data, have become free for further addition of new data to inject. This may result from the streaming out of the data filled in these buffers, or from a request from the Data Injecting Client to flush all filled buffers and to return them.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The buffers are becoming free for further addition of new data, and notified, in the same order as they were added by the Data Injecting Client.

Parameters

Table 1314

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`nrBuffers`	The number of buffers that are becoming free for further addition of new data to inject.
`pBuffersId`	Pointer to the first element of an array of cookies: each cookie identifies a previously added buffer, which is becoming free for further addition of new data to inject. The cookies are meant to simplify the buffer management task of the Data Injecting Client.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action.

Postcondition
`True`

Remarks
The way buffers are returned by means of the `OnActEnabledChanged` notification function, in “groups”, or individually, is left to the implementation, except for the case of a flush action, where all buffers are returned back to the Data Injecting Client though a single `OnActEnabledChanged` notification.

8.3.1.5.3 uhIDataInjectBuffer

Qualifiers
None.

Description
This interface of a Data Injector provides facilities to request the buffers that are managed by the platform.

Interface ID
`uuid( FB289ED1-E829-4EE0-9992-8CC691CBE5E5 )`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.1.5.3.1 GetPlatformBufferInfo

Signature

```
uhErrorCode_t GetPlatformBufferInfo(
    [out] pUInt32 pNrBuffers,
    [out] pUInt32 pSize,
    [out] pBool pBuffersSharable );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the number and the (common) size of the buffers that are managed by the platform, as well as the indication whether (platform) buffers from other Data Injector(s)/Data Extractor(s) can be used with this Data Injector.

Parameters

Table 1315

Name	Description
`pNrBuffers`	Output parameter returning the number of buffers that are managed by the Data Injector and that can be used by the Data Injecting Client to inject data.
`pSize`	Output parameter returning the size of the buffers managed by the Data Injector.
`pBuffersSharable`	Output parameter indicating whether (platform) buffers from other Data Injector(s)/Data Extractor(s) can be used with this Data Injector. It is `True` if (platform) buffers from other Data Injector(s)/Data Extractor(s) can be used with this Data Injector. It is `False` if (platform) buffers from other Data Injector(s)/Data Extractor(s) can't be used with this Data Injector.

Return Values

Standard.

Precondition

`True`

Action

None.

Postcondition

— `*pNrBuffers == dataInj.nrPlatfBuffers`

— `*pSize == dataInj.platfBuffersSize`

— `*pBuffersSharable == dataInj.platfBuffersSharable`

Remarks

— When `*pBuffersSharable` is `True`, the Data Injecting Client can use buffers from a Data Extractor and inject them back in the Data Injector to perform a "live" viewing through this extractor/injector pipeline.

--`,,```,,,,````-`-,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— If the Data injector can handle buffers from other Data Injector(s)/Data Extractor(s), then it does not use a ring buffer, and the order of injecting is no longer important.

8.3.1.5.3.2 GetPlatformBuffers

Signature

```
uhErrorCode_t GetPlatformBuffers (
    [in] UInt32 nrBuffers,
    [out, size_is(nrBuffers)] UInt32* pBuffers );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns the buffers that are managed by the platform and that can be used for injection of data into the platform by the Data Injecting Client.

It is to be noted that this function always returns the same buffers during the lifetime of the Data Injecting Logical Component Instance.

Parameters

Table 1316

Name	Description
`nrBuffers`	The number of buffers that are requested by the Data Injecting Client in order to be used for injection of data into the platform.
`pBuffers`	Output parameter returning a pointer to the first element of an array of `nrBuffers` pointers to the start address of buffers managed by the Data Injector and that can be used for injection of data into the platform by the Data Injecting Client.

Return Values

Standard.

Precondition

```
nrBuffers == dataInj.nrPlatfBuffers
```

Action

None.

Postcondition

```
For each i in {0..nrBuffers-1}

*pBuffers[i] == dataInj.platfBuffers[i]
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.2 UhlDataExtract

8.3.2.1 Concepts

8.3.2.1.1 Introduction

The function of a Data Extractor is to facilitate the extraction of any kind of data from the platform to its client.

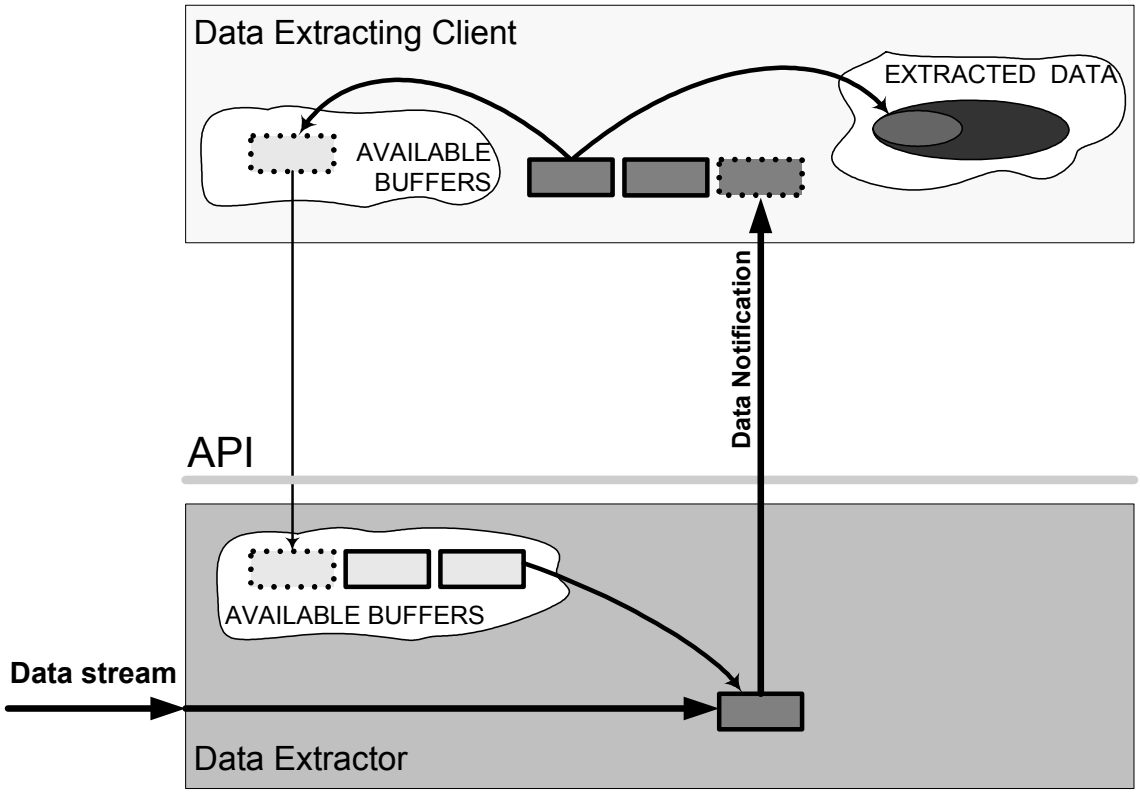


Figure 161 — Flow of data through the Data Extracting Logical Component

In Figure 161, a schematic representation of the data flow through the Data Extracting Logical Component is depicted.

The Data Extractor extracts the data from its input stream and notifies it to the client of the Logical Component via buffers: the Data Extractor adds data from its input stream to available buffers that the client provided; the filled buffer(s) are then notified by the Data Extractor to the client.

With this notification, the Data Extracting Client gets one or several filled buffers and uses them. When the client no longer needs the data in the buffer, the client can trigger the Data Extractor over the buffers that are available for further extraction of data from the platform.

The data that is notified to the client via buffers is exactly the same as the data in the input stream of the Data Extracting Logical Component. The data is also notified in the same order as it is present in the input stream of the Data Extractor.

Note that Figure 161 does not show the component that is actually managing (i.e. allocating/de-allocating) the buffers. The Data Extractor supports both buffers managed by the client and buffers managed by the platform.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.2.1.2 Data Extraction

The Data Extracting Logical Component is a generic component: it can handle the extraction of any kind of data since the Data Extractor does not perform any parsing or interpretation of the data that it notifies to its client.

This Logical Component can handle the extraction of data in the PVR context; it can also support the output of an MPEG Program Stream that the client could write to a DVD+RW.

A first consequence is that the Data Extractor does not provide any functionality to its client(s) to exclude data (or meta-data) from the extraction process: any data at the input stream of the Data Extractor is added to available buffers and notified to the Data Extracting Clients. If a client is not interested in all the data, it should remove this unwanted data itself after extraction or, if possible, it should take care to instruct an upstream component to remove the unwanted data.

Another consequence is that the Data Extractor does not report the format of the data that it extracts. Clients are typically able to know the format of the data they received, based on their knowledge of upstream Logical Components and their observable state.

Finally, the Data Extractor can't detect and therefore can't indicate to the client any error in its input stream (e.g. data-format error) since it does not parse nor interpret any of the incoming data.

8.3.2.1.2.1 Input Stream

As soon as the Data Extracting Logical Component has data available at its input, it fills one or several buffers with this data (unless otherwise specified in the Platform Instance document) if it has buffers available. If no buffer is available to the Data Extractor when new data is streaming from its input, then the data is discarded and lost. This is notified to the Data Extracting Client via an `OnOverflow` notification.

The client should consequently take care itself that there are sufficient buffers available to the Data Extracting Logical Component so that when data is available at the input of the Data Extractor, this data can be extracted and notified to the client instead of being discarded. It is therefore recommended for clients to use the notified buffers and to make them available again for filling up as soon as possible.

The Data Extractor does not provide any explicit means to its client(s) to regulate the pace at which data is extracted or to control the extraction process (e.g. start/stop functionality). The pace at which data is extracted is not determined by the pace at which the client makes known to the Data Extractor that buffers are available again for filling-in. The pace of data extraction is directly related to the pace at which data is streaming in the Data Extracting Logical Component.

8.3.2.1.2.2 Buffers

Extraction of the (streaming) data from the platform is performed via buffers.

These buffers are provided by the platform and/or by the client itself.

In principle, the Data Extracting Logical Component has a sufficient amount of buffers available that can be used for the extraction of data from the platform: an optional, but recommended, interface specified in this document provides a means to get these buffers.

Whether these buffers are allocated statically or dynamically is up to the platform instance. The number, the alignment, the location in memory, and size of these buffers as well.

In addition, if the platform allows it, the client may manage some of the buffers himself (i.e. the allocation/de-allocation). Sometimes, the client should even manage all the buffers himself: this is the case when the platform does not contribute to the buffer management at all.

When the buffers are managed only by the platform, the Data Extracting Client must add the buffers in the same order as he got them from the platform; this allows the use of ring buffers in the platform. When some of

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the buffers are managed by the Data Extracting Client itself then this constraint is not applicable. The client may then define its own buffer management.

In order to prevent extra and unnecessary copying, the buffers - added by the Data Extracting Client in order to be used for extraction of data from the input stream of the Data Extractor - can also be used to have their content written to e.g. a DVD+RW or a HDD.

To ease such kind of applications, the buffers managed by the platform must be allocated contiguous in memory; see Annex H.3 for more details.

The lifetime of the buffers that are managed by the platform itself is the same as the lifetime of the associated Logical Component instance.

The buffers that are managed by the client do not necessarily have identical sizes, while the buffers managed by the platform have identical sizes. The Data Extractor fully uses the buffer capacity to add data from the input stream, except when the last data of the current input stream is extracted. It is to be noted that the buffers are filled in the same order as they have been indicated as being available to the Data Extractor and the Data Extractor notifies the data in the same order as it was present in its input stream.

The fact that buffers have been filled with data is indicated to the client by means of an `OnDataAvailable` notification. It is possible that a single notification indicates that from this time on several buffers are filled with data originating from the platform.

A means is provided to the platform to indicate that the notified data is the last data of the current stream: this can be of interest for the processing that can possibly further happen on the extracted data.

8.3.2.1.2.3 Offset indication

In the `OnDataAvailable` notification, an offset indication is provided along with the extracted data. This offset indication represents the offset of the first byte of the first (currently) notified buffer relative to a certain position. This certain position is defined by an upstream Logical Component (e.g. an SPTS Transmuxing Logical Component) and can change over time.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

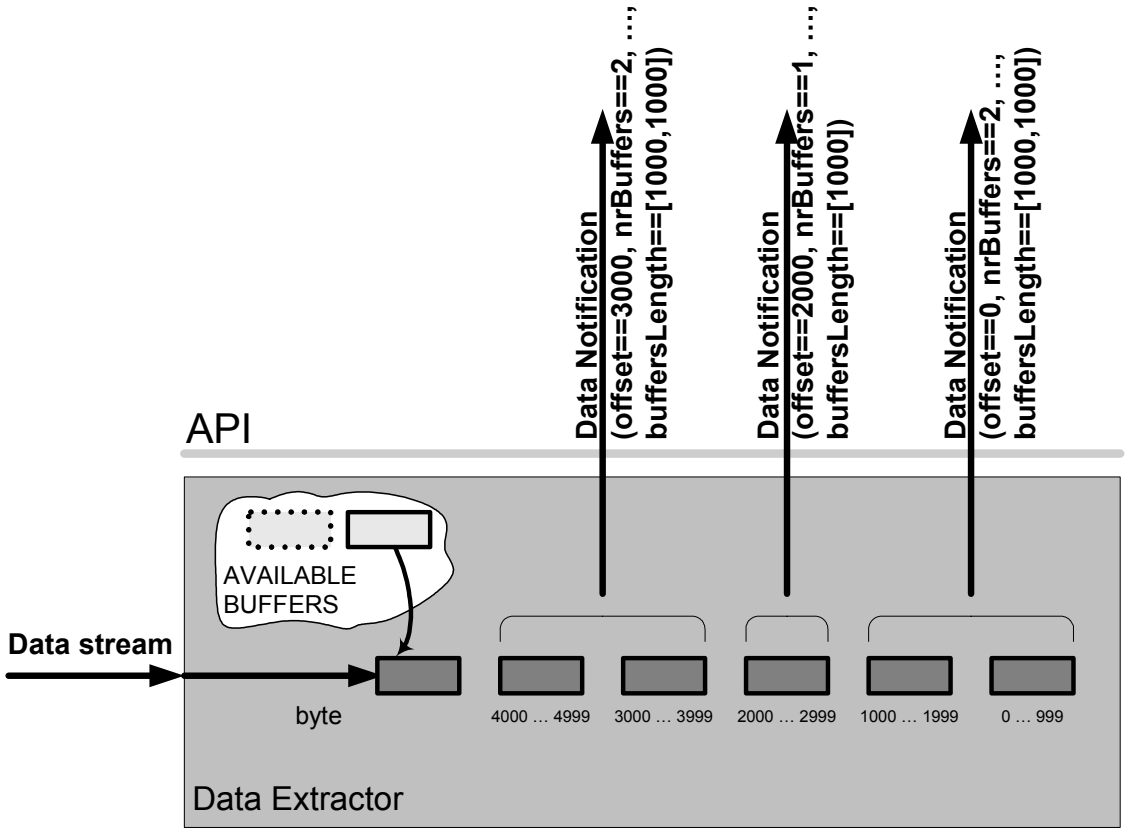


Figure 162 — Offset indication in the OnDataAvailable Notification

Figure 162 depicts how the offset indication is determined and notified: a data stream enters the Data Extractor and the data is extracted with the offset of the first data byte being equal to zero. In this example, the buffers have an identical size of 1000 bytes and the `OnDataAvailable` notification is sent with a variable number of filled buffers. Along with the offset indication, the size of the notified buffer is necessary information for the client to link meta-data (e.g. where I-Frames are situated in the stream, to further perform I-Frame only decoding) to the stream itself, if useful. The client can also use the offset indication when an overflow has happened. The offset indication gives in that case (indirectly) information over how much data has been lost.

8.3.2.2 Types & Constants

8.3.2.2.1 Public Types & Constants

8.3.2.2.1.1 Error Codes

None.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.2.2.1.2 uhDataExtract_Ntf_t

Signature

```
typedef enum _uhDataExtract_Ntf_t {
    uhDataExtract_SubscriptionChanged = 0x00000001,
    uhDataExtract_DataAvailable       = 0x00000002,
    uhDataExtract_EmptyBuffers        = 0x00000004,
    uhDataExtract_Overflow            = 0x00000008
} uhDataExtract_Ntf_t, *puhDataExtract_Ntf_t;
```

Qualifiers

— enum-element

Description

Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhIDataExtractNtf`.

Values

Table 1317

Name	Description
`uhDataExtract_SubscriptionChanged`	Represents `OnSubscriptionChanged`.
`uhDataExtract_DataAvailable`	Represents `OnDataAvailable`.
`uhDataExtract_EmptyBuffers`	Represents `OnEmptyBuffers`.
`uhDataExtract_Overflow`	Represents `OnOverflow`.

8.3.2.2.1.3 uhDataExtract_NtfSet_t

Signature

```
typedef Int32 uhDataExtract_NtfSet_t, *puhDataExtract_NtfSet_t;
```

Qualifiers

— enum-set

Description

Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type uhDataExtract`_Ntf_t`. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints

None.

8.3.2.2.1.4 uhDataExtract_AllNtfs

Signature

```
const uhDataExtract_NtfSet_t uhDataExtract_AllNtfs = 0x0000000F;
```

Qualifiers

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Defines the set of all values of type uhDataExtract_Ntf_t. This constant can be used in calls of the Subscribe and Unsubscribe functions to enable or disable all event notifications, respectively.

8.3.2.2.2 Model Types & Constants

8.3.2.2.2.1 fifoList

Signature

```
typedef struct {
    UInt32 maxNrBuffers;
    UInt32 nrBuffers;
    UInt32 buffers[maxNrBuffers];
    UInt32 buffersId[maxNrBuffers];
    UInt32 buffersLength[maxNrBuffers];
    UInt32 posEmpty;
} fifoList;
```

Qualifiers
None.

Description
Values of this type represent a sorted list of pointers to the start address of buffers that have been provided to the Data Extractor for extraction of data from its input stream.

The first element(s) of this list, if filled with (extracted) data, are always pointing to the buffer(s) that should be notified first (individually or in groups), then come in this list the pointers to the buffers that are available (i.e. empty) to contain (to-be extracted) data from the input stream of the Data Extractor.

The position of the “first” buffer available (i.e empty) for extraction of data is indicated by the posEmpty variable.

Once a new chunk of data is available at the input stream of the Data Extractor, and once this first chunk of data has been extracted, the “first” available buffer is filled with it, and can be notified to the Data Extracting Client (or notified later, if several buffers are notified together). Consequently, the previously “second” available buffer in the array becomes the “first” available in the array (the value of posEmpty is increased by one), and this is the buffer that should now be first used for the extraction of data from the input stream of the Data Extractor, and so on.

When a number of buffers are notified, they are removed from the sorted list buffers (their identifying cookie in buffersId and their size in buffersLength as well), and the value of posEmpty (as well as the value of the nrBuffers variable) is decreased by this number.

Fields

Table 1318

Name	Description
maxNrBuffers	The maximum number of pointers that the array buffers can contain.
nrBuffers	The number of pointers that the array buffers actually contains.
buffers	Array containing pointers to the buffers.
buffersId	Array containing cookies that identify each of the buffers in buffers. buffersId[i] identifies the buffer pointed by buffers[i]. The cookies are meant to simplify the buffer management task of the Data Extracting Client.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
buffersLength	Array containing with how bytes of data each buffer pointed by buffers can be filled. buffersLength[i] contains with how many bytes of data the buffer pointed by buffers[i] can be filled.
posEmpty	The index from which the buffers in buffers are available (i.e. empty) to contain (to-be extracted) data from the input stream of the Data Extractor. Buffers with indexes in {0..posEmpty-1} contain data extracted from the input stream of the Data Extractor and are meant to be further notified.

Constraints

- posEmpty ≤ nrBuffers ≤ maxNrBuffers
- for i in [nrBuffers..maxNrBuffers-1]
  - buffers[i] == 0
  - buffersId[i] == 0
  - buffersLength[i] == 0

8.3.2.3 Logical Component

8.3.2.3.1 Interface-Role Model

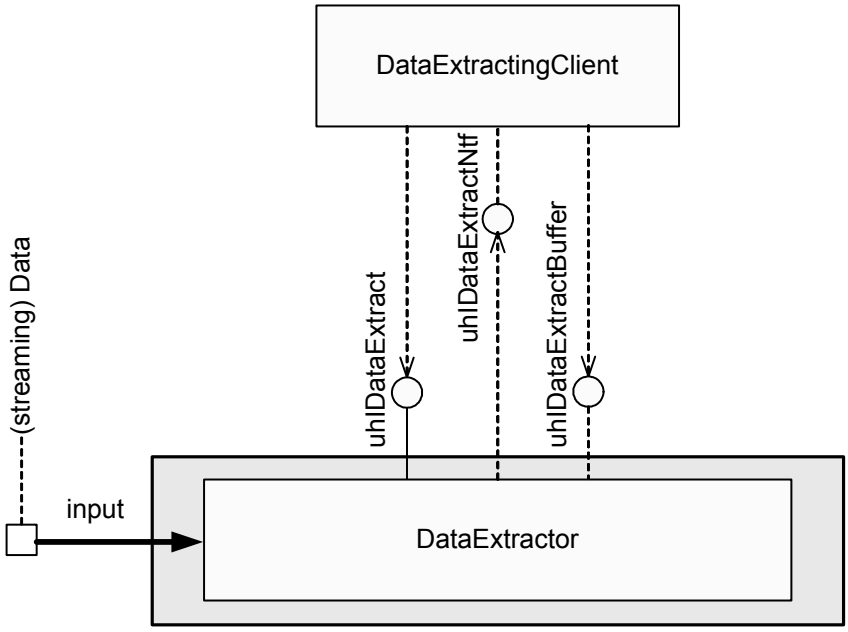


Figure 163 — Interface Role Model

The grey box in Figure 163 denotes a Data Extracting Logical Component containing a single Data Extractor role.

This role provides one mandatory (solid line) control interface (uhIDataExtract) and one optional (dotted line) control interface (uhIDataExtractBuffer).

A client can optionally provide a call-back (dotted lines) interface uhIDataExtractNtf. This allows the client to receive notifications of events that occur in the Data Extractor.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.2.3.2 Diversity

8.3.2.3.2.1 Provided Interfaces

Table 1319

Role	Interface	Presence
DataExtractor	uhIDataExtract	mandatory
DataExtractingClient	uhIDataExtractNtf	optional
DataExtractor	uhIDataExtractBuffer	optional

8.3.2.3.2.2 Configurable Items

Table 1320

Role	Attribute
DataExtractor	maxNrBuffers
	clientBuffersAllowed
	clientBuffersMinSize
	clientBuffersMaxSize
	clientBuffersAlignment
	nrPlatfBuffers
	platfBuffersSize
	platfBuffersSharable
	platfBuffers

8.3.2.3.2.3 Constraints

```
— maxNrBuffers > 0
— clientBuffersMinSize ≤ clientBuffersMaxSize
— if (!clientBuffersAllowed)
   — clientBuffersMinSize == 0
   — clientBuffersMaxSize == 0
   — clientBuffersAlignment == 0
— else
   — clientBuffersMinSize > 0
   — clientBuffersMaxSize > 0
   — clientBuffersAlignment > 0
— if nrPlatfBuffers > 0
   — platfBuffersSize > 0
— else
   — platfBuffersSharable == False
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.2.3.3 Instantiation

8.3.2.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1321

Type	Object	Multiplicity
DataExtractor	`dataExt`	1

8.3.2.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

```
— dataExt.buffersAvailToExtract.maxNrBuffers == maxNrBuffers
— dataExt.buffersAvailToExtract.nrBuffers == 0
— dataExt.buffersAvailToExtract.posEmpty == 0
— for i in [0..maxNrBuffers-1]
   — dataExt.buffersAvailToExtract.buffers[i] == 0
   — dataExt.buffersAvailToExtract.buffersId[i] == 0
   — dataExt.buffersAvailToExtract.buffersLength[i] == 0
```

8.3.2.3.4 Execution Constraints

The interfaces provided by the roles of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.3.2.4 Roles

8.3.2.4.1 DataExtractor

Signature

```
role DataExtractor {
    const UInt32      maxNrBuffers;
    const Bool        clientBuffersAllowed;
    const UInt32      clientBuffersMinSize;
    const UInt32      clientBuffersMaxSize;
    const UInt32      clientBuffersAlignment;
    const UInt32      nrPlatfBuffers;
    const UInt32      platfBuffersSize;
    const UInt32      platfBuffersSharable;
    const UInt32      platfBuffers[nrPlatfBuffers];
    fifoList          buffersAvailToExtract;
    const Bool        orderImportant;
}
```

Qualifiers

— root

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
The Data Extractor role provides means to facilitate the extraction of data from its input stream to its client(s), without any interpretation of the extracted data.

The Data Extractor role may manage (i.e. allocate/de-allocate) buffers and provide them to client(s) so that they can be used to extract data from its input stream.

Independent Attributes

Table 1322

Name	Description
`maxNrBuffers`	The maximum number of buffers that can be added by the Data Extracting Client to the set of buffers provided for extraction of data from the input stream of the Data Extractor, and that can be administrated simultaneously by the Data Extractor.
`clientBuffersAllowed`	Indication on whether the Data Extracting Client is allowed to manage buffers itself and to provide them to the Data Extractor so that they can be used to extract data from the input stream of the Data Extractor. It is `True` when the Data Extracting Client is allowed to manage buffers itself; it is `False` when the Data Extracting Client is not allowed to manage buffers itself.
`clientBuffersMinSize`	The minimum size that the buffers managed by the Data Extracting Client should have.
`clientBuffersMaxSize`	The maximum size that the buffers managed by the Data Extracting Client should have.
`clientBuffersAlignment`	The alignment that the buffers managed by the Data Extracting Client should have.
`nrPlatfBuffers`	The number of buffers that are managed by the Data Extractor and that can be provided by the Data Extracting Client to the Data Extractor to extract data from its input stream.
`platfBuffersSize`	The size of the buffers managed by the Data Extractor.
`platfBuffersSharable`	Indication on whether (platform) buffers from other Data Injector(s)/Data Extractor(s) can be used with this Data Extractor. It is `True` if (platform) buffers from other Data Injector(s)/Data Extractor(s) can be provided to this Data Extractor to extract data from its input stream. It is `False` if (platform) buffers from other Data Injector(s)/Data Extractor(s) can't be provided to this Data Extractor to extract data from its input stream.
`platfBuffers`	Array containing the start address of the buffers that are managed by the Data Extractor and that can be used to extract data from its input stream.
`buffersAvailToExtract`	Sorted list containing pointers to the start address of buffers provided for extraction of data from the input stream of the Data Extractor.

Dependent Attributes

Table 1323

Name	Description
`orderImportant`	Indication on whether the order in which the buffers are added is important. It is equal to `(!dataExt.clientBuffersAllowed && !dataExt.platfBuffersSharable)`. If it is `True` then `the Data Extracting Client must add the (platform) buffers in the same order as he got them from the platform.` `If it is False then the order in which the buffers that are managed by the platform are added, is not relevant.`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Invariants
`dataExt.buffersAvailToExtract.maxNrBuffers == maxNrBuffers`

Instantiation
A Data Extractor is always instantiated as part of a Data Extracting LC; see subclause 8.3.2.3.3.

Streaming Behavior
The Data Extractor has only one streaming input, no streaming output.

As soon as the Data Extracting Logical Component has data available at its input, it fills one or several buffers with this data (unless otherwise specified in the Platform Instance document) if it has buffers available; the filled buffer(s) are then notified by the Data Extractor to the client through the OnDataAvailable notification function. If no buffer is available to the Data Extractor when new data is streaming from its input, then the data is discarded and lost; this is notified to the client through the OnOverflow notification function.

Active Behavior
The Data Extractor role has standard notification behavior (see uhIDataExtractNtf).

Table 1324

Event	Action
Data has been extracted from the input stream of the Data Extractor, one or several buffers have been filled with this data, the Data Extractor is ready to notify the filled buffer(s).	— Let `n` be the number of buffers filled with extracted data from the input stream of the Data Extractor and that should be notified. — Let `off` be the offset indication of the first byte of the first buffer to be (currently) notified — Let `buffersId` be the array of cookies identifying each of the filled buffers to be (currently) notified — Let `buffersLength` be the array containing with how many bytes each buffer to be (currently) notified is filled — Let `eos` be the indication whether the extracted data in the buffers to be (currently) notified is the last data of the current input stream of the Data Extractor — Notify `OnDataAvailable(off,buffersId,buffersLength,eos)` — The `n` first elements from the sorted list `dataExt.buffersAvailToExtract.buffers` are removed (their identifying cookie in `dataExt.buffersAvailToExtract.buffersId` and their size in `dataExt.buffersAvailToExtract.buffersLength` as well), and the value of `dataExt.buffersAvailToExtract.posEmpty` (as well as the value of the `dataExt.buffersAvailToExtract.nrBuffers` variable) is decreased by `n`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Event	Action

There is no buffer available anymore to add data from the input stream of the Data Extractor. Data has been lost.	If `(dataExt.buffersAvailToExtract.nrBuffers>0)`
	`{`
	A number `n` of buffers filled with extracted data from the input stream of the Data Extractor have not yet been notified
	Let `off` be the offset indication of the first byte of the first buffer to be (currently) notified
	Let `buffersId` be the array of cookies identifying each of the filled buffers
	Let `buffersLength` be the array containing with how many bytes each buffer is filled
	Let `eos` be the indication whether the extracted data is the last data of the current input stream of the Data Extractor
	— Notify `OnDataAvailable(off,buffersId,buffersLength,eos)`
	— The `n` first elements from the sorted list `dataExt.buffersAvailToExtract.buffers` are removed (their identifying cookie in `dataExt.buffersAvailToExtract.buffersId` and their size in `dataExt.buffersAvailToExtract.buffersLength` as well), and the value of `dataExt.buffersAvailToExtract.posEmpty` (as well as the value of the `dataExt.buffersAvailToExtract.nrBuffers` variable) is decreased by `n`.
	`}`
	— Notify `OnOverflow()`

Execution Constraints
None.

8.3.2.4.2 DataExtractingClient

Signature
`role` DataExtractingClient `{}`

Qualifiers
— actor

Description
A Data Extracting Client represents a user of the following interfaces: uhIDataExtract and uhIDataExtractBuffer.

Execution Constraints
None.

8.3.2.5 Interfaces

8.3.2.5.1 uhIDataExtract

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This interface of a Data Extractor provides facilities to extract data from its input stream via buffers. It also provides facilities to retrieve the capabilities and the constraints of the Data Extractor with respect to buffers managed by the Data Extracting Client itself. Finally, it provides a Subscribe and an Unsubscribe function for notifications of events related to the management of buffers.

Interface ID
```
uuid( 8878CEE3-E4FA-445F-8DF7-A50D83632E50 )
```

8.3.2.5.1.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhIDataExtractNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhDataExtract_NtfSet_t notifs );
```

Qualifiers
— subscribe-function

8.3.2.5.1.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhIDataExtractNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhDataExtract_NtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

8.3.2.5.1.3 GetMaxNrBuffers

Signature
```
uhErrorCode_t GetMaxNrBuffers (
    [out] pUInt32 pMaxNrBuffers );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the maximum number of buffers that can be part of the set of buffers provided for extraction of data from the input stream of the Data Extractor.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1325

Name	Description
pMaxNrBuffers	Output parameter returning the maximum number of buffers that can be part of the set of buffers provided for extraction of data from the input stream of the Data Extractor.

Return Values
Standard.

Precondition
True

Action
None.

Postcondition
*pMaxNrBuffers == dataExt.maxNrBuffers

8.3.2.5.1.4 GetActualNrBuffers

Signature

```
uhErrorCode_t GetActualNrBuffers (
    [out] pUInt32 pActualNrBuffers );
```

Qualifiers
- synchronous
- thread-safe

Description
Returns the actual number of buffers that have been already provided to the Data Extractor for extraction of data from its input stream and that have not yet been returned back with an OnDataAvailable or an OnEmptyBuffers notification.

Parameters

Table 1326

Name	Description
pActualNrBuffers	Output parameter returning the actual number of buffers that have been already provided to the Data Extractor for extraction of data from its input stream.

Return Values
Standard.

Precondition
True

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Action
None.

Postcondition
```
*pActualNrBuffers == dataExt.buffersAvailToExtract.nrBuffers
```

8.3.2.5.1.5 AddBuffers

Signature
```
uhErrorCode_t AddBuffers (
    [in] UInt32 nrBuffers,
    [in, size_is(nrBuffers)] const UInt32* pBuffers,
    [in, size_is(nrBuffers)] const UInt32* pBuffersId,
    [in, size_is(nrBuffers)] const UInt32* pBuffersLength );
```

Qualifiers
— synchronous

— single-threaded

Description
Adds a specified number of buffers to the set of buffers provided for extraction of data from the input stream of the Data Extractor.

A Data Extracting Client should not modify any buffer that has been added, until it has received the buffer back through an OnDataAvailable or an OnEmptyBuffers notification.

If the Data Extracting Client is allowed to manage buffers,

— these can be used and therefore added successively with or simultaneously with (i.e. in the same AddBuffers function call) the buffers that are managed by the Data Extractor;

— the sizes of the buffers that are added successively (respectively, simultaneously) are not necessarily identical, a Data Extracting Client may add a buffer with size 2kb after (respectively, along with) a buffer with size 4kb for instance;

— the order in which the platform buffers are added is not relevant anymore.

Parameters

Table 1327

Name	Description
`nrBuffers`	The number of buffers that are provided for extraction of data from the input stream of the Data Extractor.
`pBuffers`	Pointer to the first element of an array of pointers to the start address of buffers provided for extraction of data from the input stream of the Data Extractor.
`pBuffersId`	Pointer to the first element of an array of cookies: each cookie identifies one of the added buffer, and each cookie is passed to the Data Extracting Client by the Data Extractor when notifying an event related to the related buffer. `pBuffersId[i]` identifies the buffer pointed by `pBuffers[i]`. The cookies are meant to simplify the buffer management task of the Data Extracting Client.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
pBuffersLength	Pointer to the first element of an array containing with how bytes of data each buffer pointed by pBuffers can be filled. pBuffersLength[i] contains with how many bytes of data the buffer pointed by pBuffers[i] can be filled.

Return Values
Standard.

Precondition
- 0 < nrBuffers ≤ dataExt.maxNrBuffers -dataExt.buffersAvailToExtract.nrBuffers
- if (dataExt.orderImportant)
  - the Data Extracting Client must add the (platform) buffers in the same order as he got them from the platform
- else
  - the order in which the buffers that are managed by the platform are added, is not relevant
- for i in {0..nrBuffers-1}
  - if pBuffers[i] is pointing to the start address of a buffer managed by the platform
    - pBuffersLength[i] == dataExt.platfBuffersSize
  - else
    - dataExt.clientBuffersMinSize ≤ pBuffersLength[i] ≤ dataExt.clientBuffersMaxSize
    - The alignment of the buffers of which pBuffers[i] is pointing to the start address is identical to dataExt.clientBuffersAlignment
  - for j in {0..nrBuffers-1}, j!=i
    - pBuffers[i] != pBuffers[j]
  - for j in {0..dataExt.buffersAvailToExtract.nrBuffers-1}
    - pBuffers[i] != dataExt.buffersAvailToExtract.buffers[j]

Action
- Modify dataExt.buffersAvailToExtract.nrBuffers
- Modify dataExt.buffersAvailToExtract.buffers
- Modify dataExt.buffersAvailToExtract.buffersId
- Modify dataExt.buffersAvailToExtract.buffersLength

Postcondition
- dataExt.buffersAvailToExtract.nrBuffers == dataExt.buffersAvailToExtract.nrBuffers'+nrBuffers
- For i in {0..nrBuffers-1}
  - dataExt.buffersAvailToExtract.buffers [dataExt.buffersAvailToExtract.nrBuffers'+i] == pBuffers[i]
  - dataExt.buffersAvailToExtract.buffersId [dataExt.buffersAvailToExtract.nrBuffers'+i] == pBuffersId[i]
  - dataExt.buffersAvailToExtract.buffersLength [dataExt.buffersAvailToExtract.nrBuffers'+i] == pBuffersLength[i]

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.2.5.1.6 RemoveEmptyBuffers

Signature
`uhErrorCode_t` RemoveEmptyBuffers `();`

Qualifiers
- asynchronous
- single-threaded

Description
Remove all empty buffers from the set of buffers provided to the Data Extractor for extraction of data from its input stream.

First, if existing, all the buffers filled with extracted data and that have not yet been notified, are notified to the Data Extracting Client through one or several OnDataAvailable notifications.

Then, all the empty buffers that have been removed are returned back to the Data Extracting Client through a single OnEmptyBuffers notification, in the same order as they were added.

This OnEmptyBuffers notification is always sent to the Data Extractor Client after calling this function, even if there is no buffer left (filled and/or empty) on the Data Extractor side.

Parameters
None.

Return Values
Standard.

Precondition
`True`

Asynchronous Action
- If `(dataExt.buffersAvailToExtract.posEmpty > 0)`
  - For all the buffers filled with extracted data and that have not yet been notified,
  - Notify (one or several) `OnDataAvailable()`
- If `(dataExt.buffersAvailToExtract.nrBuffers > 0)`
  - Modify `dataExt.buffersAvailToExtract.nrBuffers`
  - Modify `dataExt.buffersAvailToExtract.buffers`
  - Modify `dataExt.buffersAvailToExtract.buffersId`
  - Modify `dataExt.buffersAvailToExtract.buffersLength`
  - Modify `dataExt.buffersAvailToExtract.posEmpty`
- For all the empty buffers that have been removed
  - `Notify OnEmptyBuffers()`

Asynchronous Postcondition
- `dataExt.buffersAvailToExtract.nrBuffers == 0`
- `dataExt.buffersAvailToExtract.posEmpty == 0`
- For `i` in `{0..dataExt.maxNrBuffers-1}`
  - `dataExt.buffersAvailToExtract.buffers[i] == 0`
  - `dataExt.buffersAvailToExtract.buffersId[i] == 0`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— `dataExt.buffersAvailToExtract.buffersLength[i] == 0`

8.3.2.5.1.7 GetClientBufferInfo

Signature

```
uhErrorCode_t GetClientBufferInfo (
    [out] pBool pClientBuffersAllowed,
    [out] pUInt32 pMinBuffersSize,
    [out] pUInt32 pMaxBuffersSize,
    [out] pUInt32 pBuffersAlignment );
```

Qualifiers

— synchronous

— thread-safe

Description

Returns whether the Data Extracting Client is allowed to manage buffers and to provide them to the Data Extractor so that they can be used to extract data from the input stream of the Data Extractor. It also returns the minimum and the maximum size the buffers can have and their required alignment, in case the Data Extracting Client is allowed to manage buffers and to provide them.

Parameters

Table 1328

Name	Description
`pClientBuffersAllowed`	Output parameter returning whether the Data Extracting Client is allowed to manage buffers itself and to provide these buffers to the Data Extractor so that they can be used to extract data from the input stream of the Data Extractor. It is `True` when the Data Extracting Client is allowed to manage buffers itself; it is `False` when the Data Extracting Client is not allowed to manage buffers itself.
`pMinBuffersSize`	Output parameter returning the minimum size that the buffers managed by the Data Extracting Client should have. It is 0 when the Data Extracting Client is not allowed to manage buffers itself.
`pMaxBuffersSize`	Output parameter returning the maximum size that the buffers managed by the Data Extracting Client should have. It is 0 when the Data Extracting Client is not allowed to manage buffers itself.
`pBuffersAlignment`	Output parameter returning the alignment that the buffers managed by the Data Extracting Client should have. It is 0 when the Data Extracting Client is not allowed to manage buffers itself.

Return Values

Standard.

Precondition

`True`

Action

None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Postcondition
— *pClientBuffersAllowed == dataExt.clientBuffersAllowed
— *pMinBuffersSize == dataExt.clientBuffersMinSize
— *pMaxBuffersSize == dataExt.clientBuffersMaxSize
— *pBuffersAlignment == dataExt.clientBuffersAlignment

8.3.2.5.2 uhIDataExtractNtf

Qualifiers
— callback

Description
This interface should be provided by each client that wants to subscribe to notifications of events that occur in the Data Extractor. The Subscribe function of the uhIDataExtract can be used to subscribe to these events.

Interface ID
```
uuid( 3BA60340-5B0A-4D61-B90F-4EC5904BC935 )
```

8.3.2.5.2.1 OnSubscriptionChanged

Signature
```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhDataExtract_NtfSet_t notifs );
```

Qualifiers
— onsubscriptionchanged-function

8.3.2.5.2.2 OnDataAvailable

Signature
```
[async] Void OnDataAvailable (
    [in] UInt32 cookie,
    [in] UInt32 offset,
    [in] UInt32 nrBuffers,
    [in, size_is(nrBuffers)] const UInt32* pBuffersId,
    [in, size_is(nrBuffers)] const UInt32* pBuffersLength,
    [in] Bool endOfStream );
```

Qualifiers
None.

Description
This function is called by the Data Extractor whenever it has extracted data from its input stream.

The buffers are filled with data, and notified, in the same order as they were added by the Data Extracting Client. The data is notified in the same order as it was present in the input stream of the Data Extractor.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 1329

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`offset`	Offset indication representing the offset of the first byte of the first (currently) notified buffer (whose cookie is pointed to by `pBuffersId[0]`) relative to a certain position. This certain position is defined by an upstream Logical Component (e.g. an SPTS Transmuxing Logical Component) and can change over time. See subclause 8.3.2.1.2.3 and Figure 162 for more details).
`nrBuffers`	The number of buffers that contain data that has been extracted from the input stream of the Data Extractor.
`pBuffersId`	Pointer to the first element of an array of cookies: each cookie identifies a previously added buffer that now contains data that has been extracted from the input stream of the Data Extractor. The cookies are meant to simplify the buffer management task of the Data Extracting Client.
`pBuffersLength`	Pointer to the first element of an array containing with how bytes of data each buffer whose cookie is pointed to by `pBuffersId` is filled. `pBuffersLength[i]` contains with how many bytes of data the buffer whose cookie is pointed to by `pBuffersId[i]` is filled.
`endOfStream`	Indication on whether the extracted data is the last data of the current input stream of the Data Extractor. It is `True` when the extracted data is the last data of the current input stream of the Data Extractor. It is `False` otherwise. It is to be noted that `endOfStream` is `True` each time the input of the extractor becomes invalid while it was valid before.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action

Postcondition
`True`

Remarks
— The way filled buffers are returned by means of the OnDataAvailable notification function, in "groups", or individually, is left to the implementation.

— The `endOfStream` indication can be useful for some decoders that can possibly further process the extracted data. Some decoders only decode a next frame, and if their input buffer is not "full enough", e.g. at the end of a stream, the implementation may not be able to decode the last frame. With the `endOfStream` indication, the decoder can be instructed to decode the last frame as well.

8.3.2.5.2.3 OnEmptyBuffers

Signature
`[async]` `Void` OnEmptyBuffers `(`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    [in] UInt32 cookie,
    [in] UInt32 nrEmptyBuffers,
    [in, size_is(nrEmptyBuffers)] const UInt32* pEmptyBuffersId );
```

Qualifiers
None.

Description
This function is called by the Data Extractor whenever all empty buffers, which were part of the set of buffers provided for extraction of data from the input stream of the Data Extractor, have been removed.

The empty buffers are returned in the same order as they were added by the Data Extracting Client.

All empty buffers are returned back to the Data Extracting Client though a single OnEmptyBuffers notification.

Parameters

Table 1330

Name	Description
`cookie`	The value that was passed by the client at subscription time.
`nrEmptyBuffers`	The number of empty buffers that were provided to the Data Extractor for extraction of data from its input stream and that have been removed.
`pEmptyBuffersId`	Pointer to the first element of an array of cookies: each cookie identifies a previously added buffer that was provided to the Data Extractor for extraction of data from its input stream and that has been removed.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action

Postcondition
`True`

8.3.2.5.2.4 OnOverflow

Signature

```
[async] Void OnOverflow (
    [in] UInt32 cookie );
```

Qualifiers
None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
This function is called by the Data Extractor whenever there is no buffer available anymore to add data from its input stream. Data has been lost. A new OnOverflow notification can only occur once one or several buffers are added again and after that, when no buffers are available anymore to add data from the input stream.

Parameters

Table 1331

Name	Description
`cookie`	The value that was passed by the client at subscription time.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action

Postcondition
`True`

8.3.2.5.3 uhIDataExtractBuffer

Qualifiers
None.

Description
This interface of a Data Extractor provides facilities to request the buffers that are managed by the platform.

Interface ID
`uuid( 51C7CC0F-83EE-4210-8FC6-64CB3570FBAE )`

8.3.2.5.3.1 GetPlatformBufferInfo

Signature
```
uhErrorCode_t GetPlatformBufferInfo (
    [out] pUInt32 pNrBuffers,
    [out] pUInt32 pSize,
    [out] pBool pBuffersSharable );
```

Qualifiers
— synchronous

— thread-safe

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description
Returns the number and the (common) size of the buffers that are managed by the platform, as well as the indication whether (platform) buffers from other Data Injector(s)/Data Extractor(s) can be provided to this Data Extractor to extract data from its input stream.

Parameters

Table 1332

Name	Description
`pNrBuffers`	Output parameter returning the number of buffers that are managed by the Data Extractor and that can be provided by the Data Extracting Client to extract data.
`pSize`	Output parameter returning the size of the buffers managed by the Data Extractor.
`pBuffersSharable`	Output parameter indicating whether (platform) buffers from other Data Injector(s)/Data Extractor(s) can be provided to this Data Extractor to extract data from its input stream. It is `True` if (platform) buffers from other Data Injector(s)/Data Extractor(s) can be provided to this Data Extractor to extract data from its input stream. It is `False` if (platform) buffers from other Data Injector(s)/Data Extractor(s) can't be provided to this Data Extractor to extract data from its input stream.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
— `*pNrBuffers == dataExt.nrPlatfBuffers`
— `*pSize == dataExt.platfBuffersSize`
— `*pBuffersSharable == dataExt.platfBuffersSharable`

Remarks
— When `*pBuffersSharable` is `True`, the Data Extracting Client can provide buffers with extracted data from the input stream of the Data Extractor to a Data Injector to inject this data back into the platform, in order to perform a “live” viewing through this extractor/injector pipeline.

— If the Data Extractor can handle buffers from other Data Injector(s)/Data Extractor(s), then it does not use a ring buffer, and the order in which the buffers are added is no longer important.

8.3.2.5.3.2 GetPlatformBuffers

Signature
```
uhErrorCode_t GetPlatformBuffers (
    [in] UInt32 nrBuffers,
    [out, size_is(nrBuffers)] UInt32* pBuffers );
```

Qualifiers
— synchronous

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— thread-safe

Description
Returns the buffers that are managed by the Data Extractor and that can be used for extraction of data from its input stream.

It is to be noted that this function always returns the same buffers during the lifetime of the Data Extracting Logical Component Instance.

The buffers managed by the platform are allocated contiguous in memory.

Parameters

Table 1333

Name	Description
`nrBuffers`	The number of buffers that are requested in order to be used for extraction of data from the input stream of the Data Extractor.
`pBuffers`	Output parameter returning a pointer to the first element of an array of `nrBuffers` pointers to the start address of buffers managed by the Data Extractor and that can be used for extraction of data from the input stream of the Data Extractor.

Return Values
Standard.

Precondition
```
nrBuffers == dataExt.nrPlatfBuffers
```

Action
None.

Postcondition
```
— For each i in {0..nrBuffers-1}
   — pBuffers[i] == dataExt.platfBuffers[i]
```

8.3.3 UhlSptsTrMux

8.3.3.1 Concepts

8.3.3.1.1 Introduction

The SPTS Transmuxing logical component creates an SPTS out of an incoming TS. Three steps are involved in this:

— the selection of the elementary streams that must be part of the SPTS;

— the insertion of sections;

— the notification of the locations of the video frames.

The following subclauses describe these steps.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.3.1.2 Selection of the elementary streams

The TS Demultiplexer allows a client to select the elementary streams in the incoming TS that must be part of the SPTS. For example, a client can select the video elementary stream, audio elementary stream, and the elementary stream carrying the PCR of a specific program. A client may also include additional elementary streams (e.g. streams carrying data) that are part of the program. Please refer to the TS Demultiplexer specification 8.1.13 for more information.

The client should not include the elementary streams that convey the PAT and PMT. Instead, it should filter them, modify them so they match the content of the SPTS, and then insert these modified tables into the the SPTS (see the next step).

8.3.3.1.3 Insertion of sections

The client can insert sections into the SPTS using the TS Multiplexer. These sections are typically a modified PAT and a modified PMT, but the client can insert any other type of section as well. Please refer to 8.1.14 for more information.

8.3.3.1.4 Notification of the locations of the video frames

The SPTS Transmuxing logical component notifies the locations of the video frames (I-, P- and B-frames) in the SPTS. A client is typically interested in this information, as this helps the client when playing back the SPTS. The client can use the information to have random access to the SPTS (the locations in the SPTS where an I-frame begins that has the appropriate headers provide random access) and to perform trickplay.

As soon as an I-frame is detected, the client gets a notication about this. A client can then parse the headers (e.g. the sequence header) for information it is interested in. When all the frames of a GOP have been detected, the client gets a notification that provides information on the locations of all the frames in the GOP.

8.3.3.2 Types & Constants

8.3.3.2.1 Public Types & Constants

8.3.3.2.1.1 Error Codes

Signature

```
const uhErrorCode_t  UH_ERR_SPTSTRMUX_NO_FRAMEDET = 0x00000001;
```

Qualifiers

— error-codes

Description

Defines all non-standard errors that may be returned by the interfaces of the SPTS Transmuxing logical component.

Constants

Table 1334

Name	Description
UH_ERR_SPTSTRMUX_NO_FRAMEDET	This SPTS Transmuxing logical component does not support the frame detection functionality.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.3.2.1.2 uhSptsTrMux_FrameDetNtf_t

Signature
typedef enum _

```
uhSptsTrMux_FrameDetNtf_t {
    uhSptsTrMux_FrameDetSubscriptionChanged = 0x00000001,
    uhSptsTrMux_IframeDetected               = 0x00000002,
    uhSptsTrMux_GopInfo                      = 0x00000004
}

uhSptsTrMux_FrameDetNtf_t, *p

uhSptsTrMux_FrameDetNtf_t;
```

Qualifiers
— enum-element

Description
Values of this type represent notification functions. That is, there is a one-to-one correspondence between these values and the functions in the notification interface `uhISptsTrMuxFrameDetNtf`.

Values

Table 1335

Name	Description
uhSptsTrMux_FrameDetSubscriptionChanged	Represents OnSubscriptionChanged.
uhSptsTrMux_IframeDetected	Represents OnIframeDetected.
uhSptsTrMux_GopInfo	Represents OnGopInfo.

8.3.3.2.1.3 uhSptsTrMux_FrameDetNtfSet_t

Signature
typedef `Int32` uhSptsTrMux_FrameDetNtfSet_t, `*p`uhSptsTrMux_FrameDetNtfSet_t;

Qualifiers
— enum-set

Description
Values of this type represent sets of notification functions that are typically constructed by logical OR-ing of values of type

uhSptsTrMux_FrameDetNtf_t. These values are used in the `Subscribe` and `Unsubscribe` functions to specify the event notifications to be enabled or disabled.

Constraints
None.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.3.2.1.4 uhSptsTrMux_FrameDetAllNtfs

Signature

```
const uhSptsTrMux_FrameDetNtfSet_t uhSptsTrMux_FrameDetAllNtfs = 0x00000007;
```

Qualifiers

None.

Description

Defines the set of all values of type

uhSptsTrMux_FrameDetNtf_t. This constant can be used in calls of the `Subscribe` and `Unsubscribe` functions to enable or disable all event notifications, respectively.

8.3.3.2.2 Model Types & Constants

None.

8.3.3.3 Logical Component

8.3.3.3.1 Interface-Role Model

Figure 164 depicts the interface-role model for the SPTS Transmuxing logical component (the grey box in the diagram).

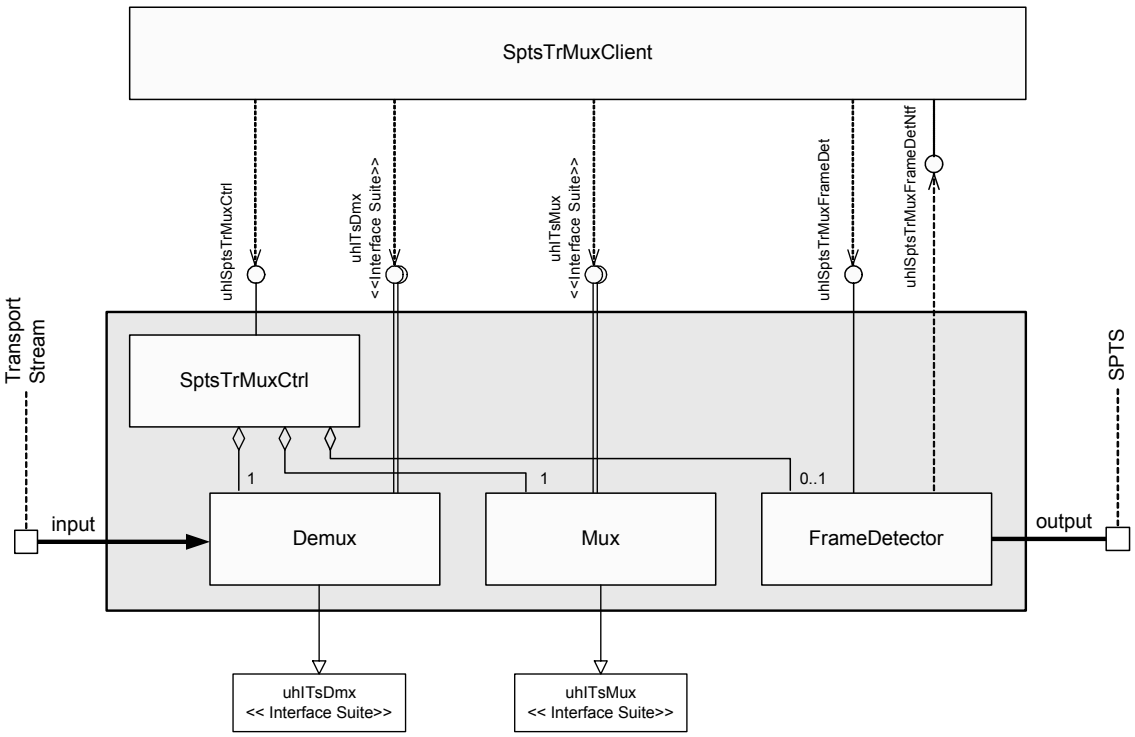


Figure 164 — Interface Role Model

The SPTS Transmuxing logical component has four roles.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The SptsTrMuxCtrl role is the root role, which provides access to the other roles through the `uhISptsTrMuxCtrl interface`.

The Demux role specializes the `uhITsDmx` interface suite (please refer to 8.1.13 for more information about the interfaces provided by the `uhITsDemux` interface suite). It allows the client to select the elementary streams that must be part of the SPTS.

The Mux role specializes the `uhITsMux` interface suite (please refer to 8.1.14 for more information about the interfaces provided by the `uhITsMux` interface suite). It allows the client to insert sections into the SPTS.

The optional FrameDetector role allows the client to be notified about the locations of video frames in the SPTS.

8.3.3.3.2 Diversity

8.3.3.3.2.1 Provided Interfaces

Table 1336

Role	Interface	Presence
`SptsTrMuxCtrl`	`uhISptsTrMuxCtrl`	mandatory
`Demux`	`uhITsDmx <<interface suite>>`	See 8.1.13
`Mux`	`uhITsMux <<interface suite>>`	See 8.1.14
`FrameDetector`	`uhISptsTrMuxFrameDet`	mandatory
`SptsTrMuxClient`	`uhISptsTrMuxFrameDetNtf`	optional

8.3.3.3.2.2 Configurable Items

Table 1337

Role	Attribute
`SptsTrMuxCtrl`	`frameDetSupp`
`Demux`	See 8.1.13
`Mux`	See 8.1.14

8.3.3.3.2.3 Constraints

None.

8.3.3.3.3 Instantiation

8.3.3.3.3.1 Objects Created

The following objects are created when the logical component is instantiated:

Table 1338

Type	Object	Multiplicity
`SptsTrMuxCtrl`	`ctrl`	1
`Demux`	`dmx`	1

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Type	Object	Multiplicity
Mux	mux	1
FrameDetector	framedet	0..1

8.3.3.3.3.2 Initial State

None.

8.3.3.3.4 Execution Constraints

The interfaces provided by an instance of this logical component may not be accessed concurrently from different threads. The only exceptions are those functions that have the qualifier "thread-safe".

8.3.3.4 Roles

8.3.3.4.1 SptsTrMuxCtrl

Signature

```
role SptsTrMuxCtrl {
    const TsDemultiplexer   dmx;
    const TsMultiplexer     mux;
    const FrameDetector     frameDet;
    const Bool              frameDetSupp;
}
```

Qualifiers

— root

Description

The SptsTrMuxCtrl role provides navigational facilities for the SptsTrMuxClient to access and interact with other roles of the SPTS Transmuxing logical component.

Independent Attributes

Table 1339

Name	Description
dmx	Reference to the TsDemultiplexer role of the Demux that is part of this SPTS Transmuxing logical component instance.
mux	Reference to the TsMultiplexer role of the Mux that is part of this SPTS Transmuxing logical component instance.
frameDet	Reference to the FrameDetector role that is part of this SPTS Transmuxing logical component instance.
frameDetSupp	Indicates whether the FrameDetector role is supported. It is True if the FrameDetector is supported. It is False otherwise.

Invariants

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Instantiation
The SptsTrMuxCtrl role is always instantiated as part of the SPTS Transmuxing logical component; see subclause 8.3.3.3.3.

Streaming Behavior
None.

Active Behavior
None.

8.3.3.4.2 Demux

Signature
role Demux **specializes** `uhITsDmx {}`

Qualifiers
— See 8.1.13 for further details.

Description
The Demux role specializes the context of `uhITsDmx` in the sense that it is being used in the SPTS transmuxing context. Since it is being used in the SPTS transmuxing context, the role of root interface of the `uhITsDmx` interface as specified in 8.1.13 is not applicable here. For further details on the `uhITsDmx` interface suite, please refer to 8.1.13.

Independent Attributes
See 8.1.13 for further details.

Dependent Attributes

See 8.1.13 for further details.

Invariants

See 8.1.13 for further details.

Instantiation
The Demux role is always instanstiated as part of the SPTS Transmuxing logical component, see subclause 8.3.3.3.3.

Streaming Behavior
See 8.1.13 for further details.

Active Behavior
See 8.1.13 for further details.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.3.4.3 Mux

Signature
role Mux **specializes** `uhITsMux {}`

Qualifiers
— See 8.1.14 for further details.

Description
The Mux role specializes the context of `uhITsMux` in the sense that it is being used in the SPTS transmuxing context. Since it is being used in the SPTS transmuxing context, the role of root interface of the `uhITsMux` interface as specified in 8.1.14 is not applicable here. For further details on the `uhITsMux` interface suite, please refer to 8.1.14.

Independent Attributes
See 8.1.14 for further details.

Dependent Attributes

See 8.1.14 for further details.

Invariants

See 8.1.14 for further details.

Instantiation
The Mux role is always instanstiated as part of a SPTS Transmuxing logical component, see subclause 8.3.3.3.3.

Streaming Behavior
See 8.1.14 for further details.

Active Behavior
See 8.1.14 for further details.

8.3.3.4.4 FrameDetector

Signature
role FrameDetector `{}`

Qualifiers
None.

Description
The FrameDetector role inspects the SPTS for the location of video frames and notifies these to its client(s).

Independent Attributes
None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Dependent Attributes

None.

Invariants

None.

Instantiation
The FrameDetector role is instantiated as part of the SPTS Transmuxing logical component if and only if `ctrl.frameDetSupp` equals `True`.

Streaming Behavior
The FrameDetector role inspects the SPTS for the location of video frames and notifies these to its client(s). This inspection does not affect the input stream and its content: the stream at the input of the FrameDetector is passed to the output unchanged.

Active Behavior
The FrameDetector role has standard notification behavior.

8.3.3.4.5 SptsTrMuxClient

Signature
role SptsTrMuxClient `{}`

Qualifiers
— actor

Description
A SptsTrMuxClient represents a client of the SPTS Transmuxing logical component. It can use all the interfaces provided by the SPTS Transmuxing logical component, and it can provide optional notification interfaces to receive events generated by the roles of the SPTS Transmuxing logical component.

8.3.3.5 Interfaces

8.3.3.5.1 SptsTrMux::uhITsDmx

Qualifiers
None.

Description
This interface of the SPTS Transmuxing logical component is a specialization of the `uhITsDmx` interface suite 8.1.13. It only specialises the context under which `uhITsDmx` is being used, i.e. the SPTS Transmuxing logical component. It is also important to note that since `uhITsDmx` is being used under the context of the SPTS Transmuxing logical component the root role of the `uhITsDmx` interface as defined in 8.1.13 is not applicable here.

Interface ID
See 8.1.13.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.3.5.2 SptsTrMux::uhITsMux

Qualifiers
None.

Description
This interface of the SPTS Transmuxing logical component is a specialization of `uhITsMux` interface suite 8.1.14. It only specialises the context under which `uhITsMux` is being used, i.e. the SPTS Transmuxing logical component. It is also important to note that since `uhITsMux` is being used under the context of the SPTS Transmuxing logical component the root role of `uhITsMux` as defined in 8.1.14 is not applicable here.

Interface ID
See 8.1.14.

8.3.3.5.3 uhISptsTrMuxCtrl

Qualifiers
— root

Description
This interface allows the client to navigate to the other roles of the SPTS Transmuxing logical component.

Interface ID
`uuid( 9993AB43-0B0D-4FB5-A8A4-554E196E260A )`

8.3.3.5.3.1 GetDemux

Signature
```
uhErrorCode_t GetDemux (
    [out] uhITsDmx **ppDemux );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the `uhITsDmx` interface of the `Demux`. Clients can use this interface to further exercise control on the `Demux`.

Parameters

Table 1340

Name	Description
`ppDemux`	Output parameter returning an interface pointer to `uhITsDmx`.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*ppDemux contains a pointer to the uhITsDmx interface of ctrl.dmx`

8.3.3.5.3.2 GetMux

Signature

```
uhErrorCode_t GetMux (
    [out] uhITsMux **ppMux );
```

Qualifiers
— synchronous

— thread-safe

Description
Returns the `uhITsMux` interface of the `Mux`. Clients can use this interface to further exercise control on the `Mux`.

Parameters

Table 1341

Name	Description
`ppMux`	Output parameter returning an interface pointer to `uhITsMux`.

Return Values
Standard.

Precondition
`True`

Action
None.

Postcondition
`*ppMux contains a pointer to the uhITsMux interface of ctrl.mux`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.3.5.3.3 GetFrameDetector

Signature

```
uhErrorCode_t GetFrameDetector(
    [out] uhISptsTrMuxFrameDet **ppFrameDetector );
```

Qualifiers

— synchronous

— thread-safe

Description

— Returns the

`uhISptsTrMuxFrameDet` interface of the `FrameDetector` if available.

Parameters

Table 1342

Name	Description
`ppFrameDetector`	— Output parameter returning an interface pointer to `uhISptsTrMuxFrameDet` if available.

Return Values

Only non-standard error values are listed here.

Table 1343

Value	Description
`UH_ERR_SPTSTRMUX_NO_FRAMEDET`	This SPTS Transmuxing logical component does not support the frame detection functionality.

Precondition

`True`

Action

None.

Postcondition

— `if ctrl.frameDetSupp`
  — `*ppFrameDetector contains a pointer to the`
  — `uhISptsTrMuxFrameDet interface of ctrl.frameDet`
— `else`
  — `*ppFrameDetector == NULL`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.3.5.4 uhISptsTrMuxFrameDet

Qualifiers
None.

Description
This interface of FrameDetector provides a subscription and an unsubscription function on notifications of events related to the detection of I-frames and to the detection of all frames of a GOP, in the SPTS.

Interface ID
```
uuid( E838EE55-CF4E-49EA-A552-1813C3E67F7B )
```

8.3.3.5.4.1 Subscribe

Signature
```
uhErrorCode_t Subscribe (
    [in] uhISptsTrMuxFrameDetNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhSptsTrMux_FrameDetNtfSet_t notifs );
```

Qualifiers
— subscribe-function

8.3.3.5.4.2 Unsubscribe

Signature
```
uhErrorCode_t Unsubscribe (
    [in] uhISptsTrMuxFrameDetNtf *pINotify,
    [in] UInt32 cookie,
    [in] uhSptsTrMux_FrameDetNtfSet_t notifs );
```

Qualifiers
— unsubscribe-function

8.3.3.5.5 uhISptsTrMuxFrameDetNtf

Qualifiers
— callback

Description
This interface should be provided by each SptsTrMuxClient that wants to subscribe to notifications of events related to the detection of I-frames and to the detection of all frames of a GOP, in the SPTS.

Interface ID
```
uuid( 8ECBC586-17CC-411F-9156-E1E19747F8E8 )
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.3.5.5.1 OnSubscriptionChanged

Signature

```
[async] Void OnSubscriptionChanged (
    [in] UInt32 cookie,
    [in] uhSptsTrMux_FrameDetNtfSet_t notifs );
```

Qualifiers

— onsubscriptionchanged-function

8.3.3.5.5.2 OnIframeDetected

Signature

```
[async] Void OnIframeDetected (
    [in] UInt32 cookie,
    [in] UInt32 offsetHigh,
    [in] UInt32 offsetLow,
    [in] UInt32 length,
    [in] Bool seqHeaderPresent,
    [in] Bool ptsPresent,
    [in] uhTimeStamp_t pts,
    [in] UInt32 tapecount );
```

Qualifiers

None.

Description

This function is called by FrameDetector whenever it detects an I-frame in the incoming SPTS. Information over the location in the SPTS where the detected I-frame begins is provided along with the notification.

Parameters

Table 1344

Name	Description
cookie	The value that was passed by the client at subscription time.
`offsetHigh`	Bits 63 – 32 of an offset indication representing the location of the first TS packet containing the detected I-frame or any of the preceding header layers (if present). This is a byte offset starting at 0 when the SPTS Transmuxing logical component is instantiated. The offset is continuously increasing during the lifetime of a SPTS Transmuxing logical component. For example, when a signal becomes temporarily unavailable/invalid, the offset is not reset to 0. If the offset was reset to 0, then there would be the possibility of race conditions.
`offsetLow`	Bits 31 – 0 of an offset indication representing the location of the first TS packet containing the detected I-frame or any of the preceding header layers (if present).
`length`	The length of the detected I-frame in bytes, including any preceding header layers. `offset`+`length-1` points to the last byte of the last TS packet containing a part of this I-frame.
`seqHeaderPresent`	Indication on whether the I-frame is preceded by a sequence header (making it a valid access point) or not. It is `True` if the I-frame is preceded by a sequence header. It is `False`, otherwise.
`ptsPresent`	Value that indicates whether `pts` contains a valid PTS. Is `True` if `pts` contains a valid PTS. It is `False`, otherwise.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
`pts`	The presentation time stamp of the I-frame, measured in 90 kHz cycles. If `pts` is `False`, the content of `pts` is irrelevant.
`tapecount`	Tapecount indication, in unit of 10 ms. The tapecount represents the presentation time of the I-frame, only without the discontinuities that may be present in the PTS: the tapecount "stops" when the FrameDetector has no input, and continues (without a discontinuity) when the FrameDetector has input again. The tapecount starts at 0 when the SPTS Transmuxing logical component is instantiated.

Return Values
None.

Precondition
`True`

Action
— Any callback-compliant action

Postcondition
`True`

Remarks
— In the segment indicated by `offset` and `length`, there may also be TS packets that are not related to the detected I-frame (e.g. audio packets, or TS packets containing sections inserted by the client in the Mux).

— The SPTS Transmuxing Client(s) can detect PTS discontinuities by comparing PTS values of consecutive I-frames. It two consecutive PTS values differ too much, there must have been a PTS discontinuity.

— The tapecount indication is provided because it may be difficult for the SPTS Transmuxing Client(s) to calculate positions in the stream based on the given PTS because of possible PTS discontinuities. The tapecount will not have discontinuities.

8.3.3.5.5.3 OnGopInfo

Signature
```
[async] Void OnGopInfo (
    [in] UInt32 cookie,
    [in] UInt32 iFrameOffsetHigh,
    [in] UInt32 iFrameOffsetLow,
    [in] UInt8 nrPframes,
    [in] UInt8 nrBframes,
    [in, size_is(nrPframes)] const pUInt32 pPframesRelOffset,
    [in, size_is(nrBframes)] const pUInt32 pBframesRelOffset );
```

Qualifiers
None.

Description
This function is called by FrameDetector whenever it has detected all the frames of a GOP. Information over the location of all the frames in the GOP (I-, P-, and B-frames) is provided along with the notification.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE For the purpose of this function, it is defined that a GOP always starts with an I-frame and ends with the last frame before the next I-frame.

Parameters

Table 1345

Name	Description
cookie	The value that was passed by the client at subscription time.
iFrameOffsetHigh	Bits 63 – 32 of an offset indication representing the location of the first TS packet containing the detected I-frame or any of the preceding header layers (if present). This is a byte offset starting at 0 when the SPTS Transmuxing logical component is instantiated. The offset is continuously increasing during the lifetime of a SPTS Transmuxing logical component. For example, when a signal becomes temporarily unavailable/invalid, the offset is not reset to 0. If the offset was reset to 0, then there would be the possibility of race conditions.
iFrameOffsetLow	Bits 31 – 0 of an offset indication representing the location of the first TS packet containing the detected I-frame or any of the preceding header layers (if present).
nrPframees	Number of P-frames in the detected GOP.
nrBframes	Number of B-frames in the detected GOP.
pPframesRelOffset	Pointer to the first element of an array of offset indications representing the locations of the first TS packet containing each P-frame of the detected GOP. This offset indication is relative to the offset indication of the I-frame in iFrameOffsetHigh and iFrameOffsetLow.
pBframesRelOffset	Pointer to the first element of an array of offset indications representing the locations of the first TS packet containing each B-frame of the detected GOP. This offset indication is relative to the offset indication of the I-frame in iFrameOffsetHigh and iFrameOffsetLow.

Return Values
None.

Precondition
True

Action
— Any callback-compliant action

Postcondition
True

Remarks
The offset indication of the I-frame in iFrameOffsetHigh and iFrameOffsetLow is the same offset indication as used in the OnIframeDetected notification that notifies the same I-frame but individually.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9 Security interface suites

9.1 IPMP interface suites

9.1.1 Introduction

The IPMP interfaces of Multimedia API can be divided into two groups: (i) trust management interfaces and (ii) tool interfaces. The trust management interfaces are to support tool's functionality by accessing secured information. The tool interfaces are to perform a variety of IPMP functions based on IPMP information.

Trust management interfaces play role of accessing secured or governed information such as key, signature, certificate, license and domain, while IPMP tool are processing under IPMP terminal. And tool interfaces are means to process secured or governed contents and to manage tools. Figure 165 — Classification diagram of IPMP interfaces, shows the classification diagram of IPMP interfaces according to the description above.

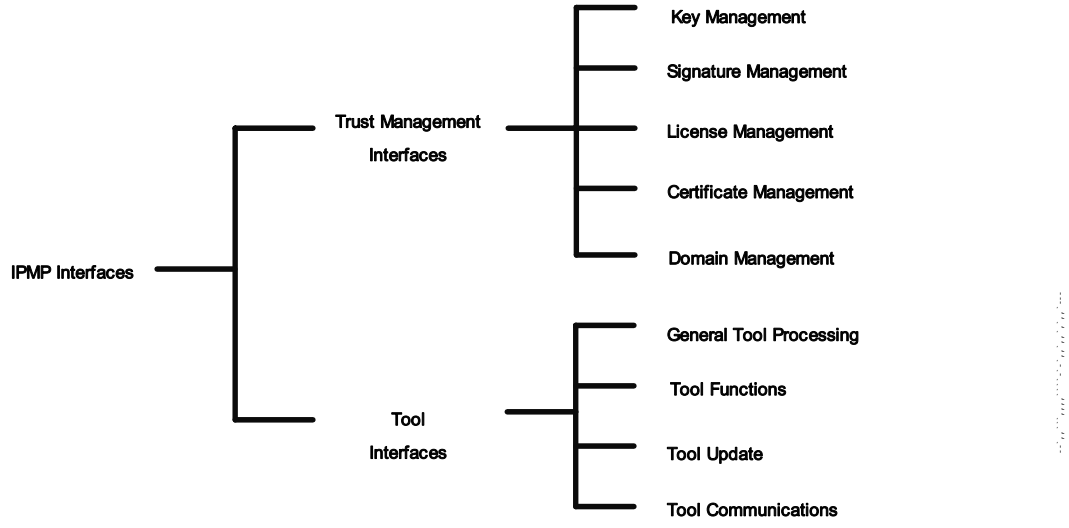


Figure 165 — Classification diagram of IPMP interfaces

9.1.2 M3W view of IPMP Service

All IPMP Services in M3W should obey the realization technology defined in ISO/IEC 23004-3.

9.1.2.1 IPMP tool Service Instance Creation

According to the IPMP information, a required IPMP tools should be retrieved and then executed in IPMP Terminal. If there is no proper tool, a tool manager of IPMP Terminal should get missing tools from location information of the IPMP information.

Also, Service Manager in M3W component model can help to find out the appropriate tool services. In Figure 166 — An example of IPMP processing, it assumes that IPMP Terminal has all tools and do not need to find the missing tools. IPMP Tool Manger of IPMP Terminal requests M3W's service manger to get the service instance of tools. And IPMP tool manger requests each tools to start `operateTool()`. This could be done by using `Start()` method in `RcIService` interface that is implemented by the IPMP Tool service. During executing IPMP tool service, whenever and wherever IPMP processing is required, tool performs its own IPMP functions.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

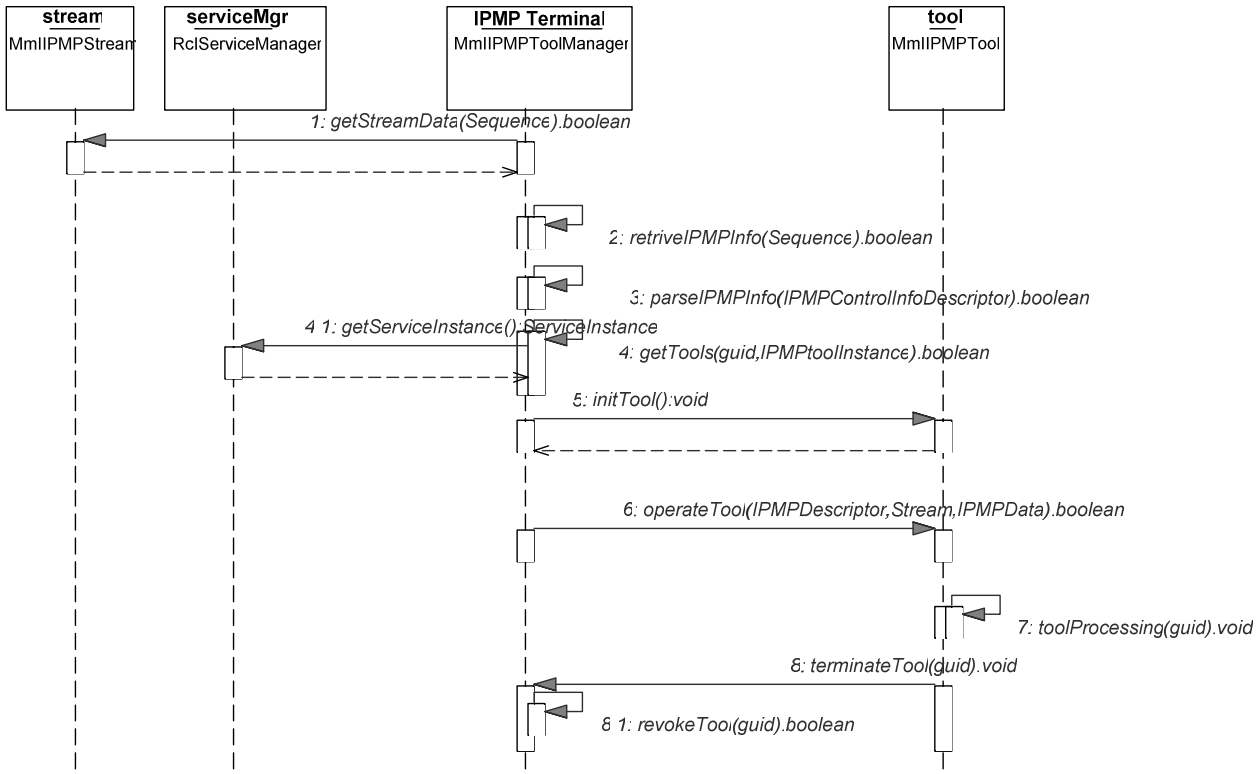


Figure 166 — An example of IPMP processing

9.1.3 IPMP Interfaces

9.1.3.1 Trust Management Interface

In trust management interfaces, it needs to consider how to access secured or governed information in processing multimedia data or performing IPMP tools. In IPMP, governed information can be described by key, signature, certificate, and license. These types of information are used in most security systems. And related functions were also defined in other APIs such as OpenSSL and PlatformSDK. In this section, we focus only on functionalities for IPMP processing.

9.1.3.1.1 Key Management Operation

Key management operation is essential for IPMP functions such as encryption, decryption, watermarking, authentication, and so on. These functions are used to process multimedia data in accordance with the key information. Following offers type definitions and several APIs for key management:

```
typedef URI ByteArray ;
typedef struct KeyInfo { // Datatype for Key Information
   URI keyName;
   URI MgmtData;
};
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
interface MmIKeyManagement { 6912b8aa-0b2a-4f94-922b-7ea1af715b86 } {
    Bool getKey(in URI keyURI);
    Bool generateKey(in KeyInfo keyInfo);
    Bool getKeyInfo(Void);
    Bool setKeyInfo(in KeyInfo keyInfo);
    Bool exchangeKey(out KeyInstance inputKey);
    Void destroyKey(in UUID keyID);
    Bool exportKey(in URI targetURI);
    Bool importKey(in URI sourceURI);
};
```

- `getKey`

Parameters	URI keyURI
Semantics	This function acquires keys for IPMP tools or IPMP terminals. It is called only when key is required for IPMP processing.

- `generateKey`

Parameters	KeyInfo keyInfo(optional)
Semantics	This function generates a key according to key parameters which can be inserted as an input parameters or pre-inserted by the `setKeyInfo` function.

- `getKeyInfo`

Parameters	Void
Semantics	This function is called when the parameters of the current key are required. It returns key parameters as a KeyInfo. These parameters can be utilized for checking key information or validating keys in IPMP terminal

- `setKeyInfo`

Parameters	KeyInfo keyInfo
Semantics	This function is for setting or changing key parameters. The modified parameters can be implied when a key is generated or utilized.

- `exchangeKey`

Parameters	KeyInstance inputKey
Semantics	This function is utilized for changing the stored key into input key in IPMP terminal

- `destroyKey`

Parameters	guid keyID
Semantics	This function is utilized for invalidating stored keys in IPMP terminal

- `exportKey`

Parameters	URI targetURI
Semantics	This function is utilized for exporting stored keys in IPMP terminal to outside device indicated as targetURI

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- importKey

Parameters	URI sourceURI
Semantics	This function is utilized for importing keys from sourceURI.

9.1.3.1.2 Signature Management Operation

A *digital signature* means a value generated from the application of a private key to a message via a cryptographic algorithm such that it has the properties of integrity, message authentication and/or signer authentication. In IPMP, the signature is important to represent immunity of IPMP properties. Followings are basic signature functions to IPMP.

```
typedef ByteArray SignatureValueType;
typedef struct Signature {
    SignedInfo signedInfo;
    SignatureValueType signatureValue;
    KeyInfo keyInfo;
    sequece <object> sigObject;
};

interface MmISignatureManagement { 4de8b4ef-454d-4427-9c01-2a7950e63204 } {
    Bool generateSignature(in SignatureInfo sigInfo);
    Bool getSignature(in URI sourceUri);
    Bool verifySignature(in Signature signature);
    Int32 querySignatureAlgorithms( out stringArray AlgorithmList);
}
```

- querySignatureAlgorithms

Parameters	out stringArray AlgorithmList
Semantics	This function is called when a Registration Authority issue standard identifiers for established algorithms.

- generateSignature

Parameters	SignatureInfo sigInfo
Semantics	This function is called when a source device or a user requests to create a digital signature. It returns a signature based on input information.

- getSignature

Parameters	URI sourceUri
Semantics	This function is called when a digital signature is required to a tool for authentication or certification.

- verifySignature

Parameters	Signature signature
Semantics	This function is used for verifying a digital signature. It returns SUCCESS or TRUE if the signature is valid; otherwise it returns FAIL or FALSE.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.1.3.1.3 License Management Operation

License is a representation of right expression for digital contents. In DRM system, usage rule and copyright are written in a license with key information. Figure 167 — License management, shows a procedure of license management.

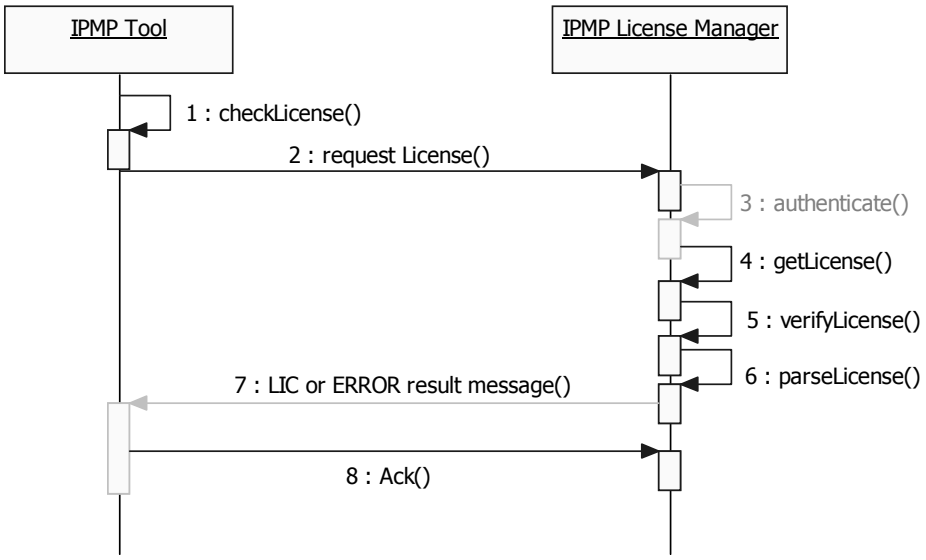


Figure 167 — License management

```
/**
*  This interface is for License management operation.
*/
typedef ByteArray LicenseInstance ;
typedef struct LicenseInfo {
    sequence <ByteArray> licenseData;
    URI licenseRef;
};


Interface MmILicenseManager { 769072c1-4b1c-4fed-8a5d-07a1114c900b } {
    Bool checkLicense(in URI sourceURI);
    Bool getLicense(in URI sourceURI);
    Void requestLicense(in URI targetURI);
    Bool verifyLicense(in LicenseInstance license);
    Void requestLicenseUpdating(in URI targetURI,
    in LicenseInfoDescriptor licenseUpdateInfo);
    Bool updateLicense(in LicenseInfoDescriptor licenseUpdateInfo);
    Void requestLicenserevocation (in URI targetURI,
    in LicenseInfoDescriptor licenseUpdateInfo);
    Bool revokeLicense(in LicenseInstance license);
    Bool queryLicenseAuthorization(in ByteArray license,
    Stream resource, ByteArray licenseInfo);
}
```

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.1.3.1.3.1 Access License

For an MPEG data, IPMP terminal should check the existence of licenses related to the data. If a license is required, the IPMP terminal requests the license for processing data. The IPMP terminal verifies and parses the licenses for usage information (i.e., usage rule and copyright etc.) about MPEG data. Followings are functions that are required to access licenses:

- `checkLicense`

Parameters	Void
Semantics	This function checks the existence of licenses for multimedia data.

- `requestLicense`

Parameters	URI targetURI
Semantics	This function requests a license for target device or module.

- `getLicense`

Parameters	URI sourceURI
Semantics	This function is called for downloading or transmitting a license from sourceURI. It returns a positive value when downloading or transmission is successful; otherwise it returns a negative value.

- `verifyLicense`

Parameters	`LicenseInstance` license
Semantics	This function is used for verifying a given license. It returns a positive value when the license is valid; otherwise it returns a negative value.

- `parseLicense`

Parameters	`LicenseInstance` license
Semantics	This function parses a license.

- `queryLicenseAuthorization`

Parameters	In ByteArray license, Stream resource, ByteArray licenseInfo
Semantics	This function is called to request to check for the existence of an authorization proof for an authorization request

9.1.3.1.3.2 Update License

If a license for an MPEG data becomes invalid, a user should update the license to access the data properly. To update a license, a user requests the updating of license to license server or license authortity. After the license is permitted to be updated, the user can update the license. Following functions are required for this license updating procedure. Note that the same functions used in previous procedure are skipped.

- `requestLicenseUpdating`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters	LicenseInfoDescriptor licenseUpdateInfo
Semantics	This function requests updating a license. In a license, data updated is determined from update information, licenseUpdateInfo.

- `updateLicense`

Parameters	`LicenseInstance` licenseUpdate
Semantics	This function updates the stored license into the specified one. It returns the updated license instance.

9.1.3.1.3.3 Revoke License

If a license for an MPEG data becomes invalid and a user does not need it anymore, the license should be revoked. Similar to other processing, to revoke a license, the request is sent first, and revocation is followed. The revocation information may be stored in a revocation list. Following functions are required ones for this license revoking procedure.

- `requestLicenserevocation`

Parameters	LicenseInfoDescriptor licenseInfo
Semantics	This function requests revoking a license.

- `revokeLicense`

Parameters	LicenseInfoDescriptor licenseInfo
Semantics	This function revokes a specified license. If there is a revocation list, information of the revoked license is added into the list.

9.1.3.1.4 Certificate Management Operation

Key is very important information, but it requires its secured transmitting methods. Certificate is one of the most popular methods to manage key and related information. Figure 168 — Certificate management: issuing, updating, and revoking and Figure 169 — Authentication using a certificate show procedures to manage a certificate and to authenticate using a certificate, respectively. For authentication using certificates, it is required to issue a certificate, and authentication can be done using the issued certificate. For managing the certificate, a method to update or revoke is also needed.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

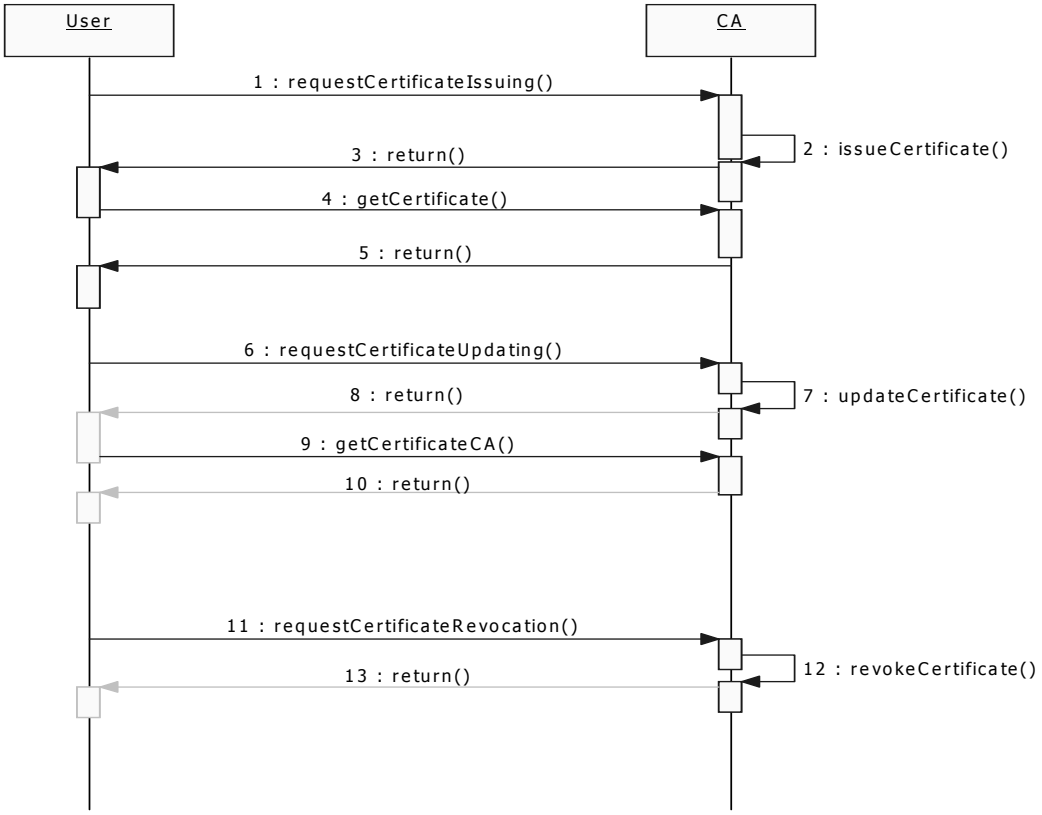


Figure 168 — Certificate management: issuing, updating, and revoking

```
/**
* Data type for Cerfificate management
**/
typedef struct CertificateInfo
{
    UInt16 CertificateType;
    URI CertificateMethod;
    sequence <UInt8> CertificateInfo_Data ;
} ;


/**
* interface MmICertificateCA : interface for certificat management(CA).
*
**/

interface MmICertificateCA { b6a8feed-5e09-4ac1-884a-f2459234045d } {
    Bool issueCertificate(in CertificateInfo certificateInfo);
    Bool updateCertificate(in CertificateInfo certificateUpdateInfo);
    Bool revokeCertificate(in CertificateInfo certificateRevocation);
}
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.1.3.1.4.1 Issue Certificate

To utilize a certificate, a user (or device) should request to issue one, then a certificate authority (CA) issues a certificate based on the information from the request. If the certificate is successfully issued, the user could get the certificate from the CA. Following functions are required for this certificate issuing procedure:

- `requestCertificateIssuing`

Parameters	URI caURI CertificateInfo certificateInfo
Semantics	This function is called when a user (or device) requests issuing a certificate for authentication.

- `issueCertificate`

Parameters	CertificateInfo certificateInfo
Semantics	This function is called after a user (or device) requests issuing a certificate. It returns the issued certificate information.

- `getCertificateCA`

Parameters	URI caURI
Semantics	This function is called after a user (or device) requests issuing a certificate. It returns the issued certificate.

9.1.3.1.4.2 Update Certificate

When a certificate stored in IPMP supported devices is not valid, a user should be able to update the certificate if necessary. The same as the certificate issuing, the request to update a certificate is held from user, then the updating of a certificate is performed in CA. Finally, the results are sent to the requester. Following functions are required for this certificate updating procedure. Note that the same functions used in previous procedure are skipped.

- `requestCertificateUpdating`

Parameters	URI caURI CertificateInfo certificateUpdateInfo
Semantics	This function is called when a user (or device) requests updating a certificate for authentication.

- `updateCertificate`

Parameters	CertificateInfo certificateUpdateInfo
Semantics	This function is called after a user (or device) requests updating a certificate. It returns the updated certificate information. In a certificate data updated is determined from update information, certificateUpdateInfo.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.1.3.1.4.3 Revoke Certificate

When a certificate stored in IPMP supported devices is not valid, if a user does not want to use it anymore the certificate should be revoked. Following functions are required for revoking this certificate revoking procedure:

```
/**
*  This interface is for Certificating on Trust management service.
*/
typedef ByteArray CertificateInstance ;


interface MmICertificateUser { edb94079-6747-4c14-acaf-acee858b2c7a } {
   Void requestCertificateIssuing(in URI caURI,
   in CertificateInfo certificateInfo);
   Bool getCertificateCA(in URI caURI);
   Void requestCertificateUpdating(in URI caURI,
   in CertificateInfo certificateUpdateInfo);
   Void requestCertificateRevocation(in URI caURI,
   in CertificateInfo certificateRevocation);
   Bool importCertificate(in URI sourceURI);
   Bool exportCertificate(in URI targetURI);
   Void initAuthentication(Void);
   Void requestCertificate(in URI targetURI);
   Bool getCertificate(in URI targetURI);
   Bool validateCertificate(in CertificateInstance certificate);
}
```

- `requestCertificateRevocation`

Parameters	URI caURI CertificateInfo certificateRevocation
Semantics	This function is called when a user (or device) requests revoking a certificate.

- `revokeCertificate`

Parameters	CertificateInfo certificateRevocation
Semantics	This function makes the certificate revoked.

9.1.3.1.4.4 Import/Export Certificate

A certificate can be moved from an IPMP device to another. This movement can be supported by importing and exporting functions as following:

- `importCertificate`

Parameters	URI sourceURI
Semantics	This function is utilized for importing keys from sourceURI.

- `exportCertificate`

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters	URI targetURI
Semantics	This function is utilized for exporting keys to the device targetURI.

9.1.3.1.4.5 Authentication using a Certificate

When a device or a component has its certificate, it is ready for authentication. If a device A wants to authenticate a device B, the device A requests the device B's certificate and validates it. Figure 169 — Authentication using a certificate shows a procedure of authentication using a certificate.

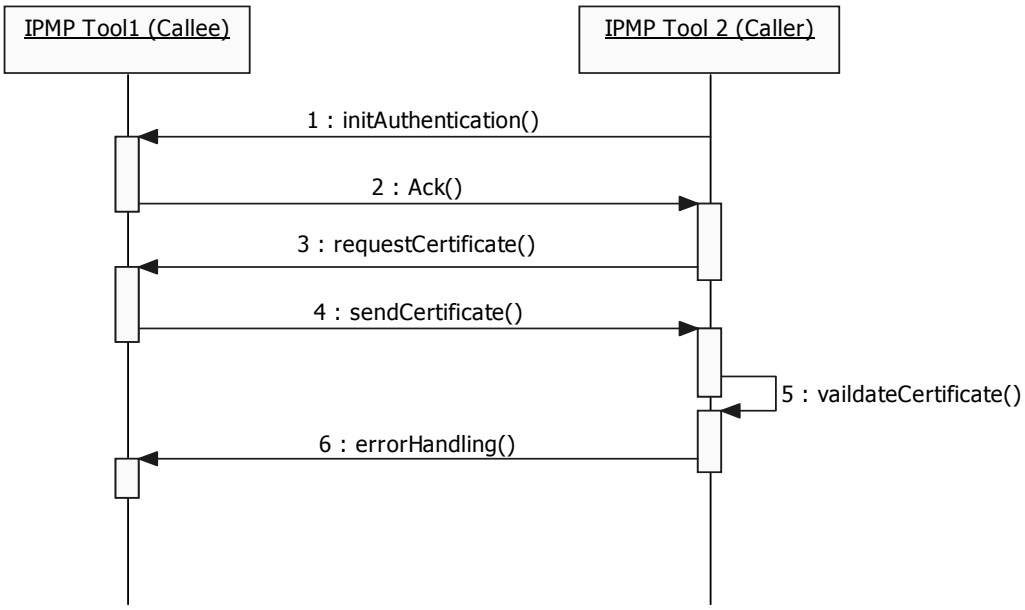


Figure 169 — Authentication using a certificate

- `initAuthentication`

Parameters	void
Semantics	This function initializes an authentication module.

- `requestCertificate`

Parameters	URI targetURI
Semantics	This function requests a certificate to a given tool (or a device).

- `getCertificate`

Parameters	URI targetURI
Semantics	This function is called for downloading or transmitting a certificate from targetURI. It returns a positive value when downloading or transmission is successful; otherwise it

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	returns a negative value.

- validateCertificate

Parameters	CertificateInstance certificate
Semantics	This function is used for verifying a given certificate. It returns a positive value when the certificate is valid; otherwise it returns a negative value.

9.1.3.1.5 Domain Management Interface

Domains are groups of devices or users sharing some common properties and IPMP Domain Management Interface functions are defined to manipulate domain elements such as domain information, device information and so on. The domain management interfaces are composed of 5 kinds of functions: Add Domain, Revoke Domain, Update Domain, Subscribe Domain and Unsubscribe Domain. Specifically MmIDomainManager is defined as follow:

```
Interface MmIDomainManager { 6954dfc1-ec5b-4ca9-a2d0-890cd44c82b2 }
{
Bool requestNewDomain(in DomainInfo domainInfo);
Bool requestRevokeDomain (in DomainInfo domainInfo);
Bool requestAuthenticate(in DomainInfo domainInfo)
Bool requestupdateDomain(in DomainInfo domainInfo);
Bool createDomainInfo(in DomainInfo domainInfo);
Bool deleteDomainInfo(in DomainInfo domainInfo);
Bool createDeviceInfo(in DomainInfo domainInfo);
Bool deleteDeviceInfo(in DomainInfo domainInfo);
Bool verifyDomainInfo(in DomainInfo domainInfo)
Bool updateDomain(in DomainInfo domainInfo,
in DomainInfo newDomainInfo);
}
```

Before a domain administrator or a device uses a function of domain management, it should be verified by Authenticator whether the information based on Certificate is verified or not.

9.1.3.1.5.1 Add Domain

A domain in which a user or a device participates should be generated beforehand. To add new domain, a domain manager requests authentication of domain information such as administrator to Authenticator. Once Authenticator receives the required information from the domain manager, the Authenticator verifies the user which will be the domain administrator. If the information is valid, the domain manager creates new domain and returns the result to the initiator.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

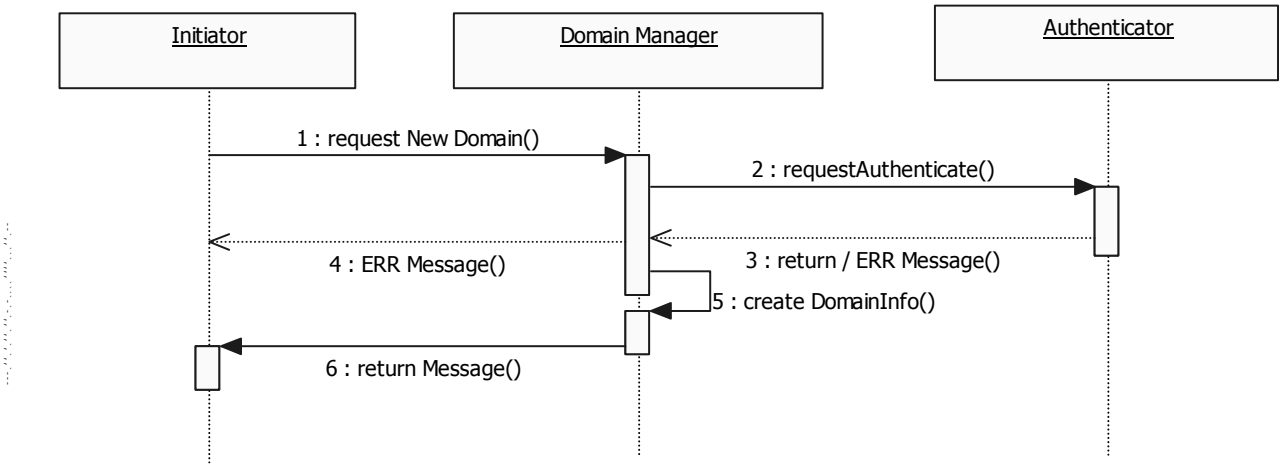


Figure 170 — Add New Domain

- `requestNewDomain`

Parameters	`DomainInfo domainInfo`
Semantics	This function requests a new domain added.

- `requestAuthenticate`

Parameters	`DomainInfo domainInfo`
Semantics	This function verifies immunity of input information such as administrator. If the information is valid, it returns `true`, otherwise, `false` is returned.

- `createDomainInfo`

Parameters	`DomainInfo domainInfo`
Semantics	This function adds new domain associated with new Domain. It returns `true` if there is no error occurred, otherwise, `false` is returned.

9.1.3.1.5.2 Revoke Domain

The domain information should be removed from a domain manager(DM), when the domain is not required any longer. To remove information of a domain, an administrator requests removing Domain to DM. Once the DM receives domain removal message from a user device, the message is transmitted to Authenticator for verifying. And then DM revokes the domain information and it returns the result to a user device.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

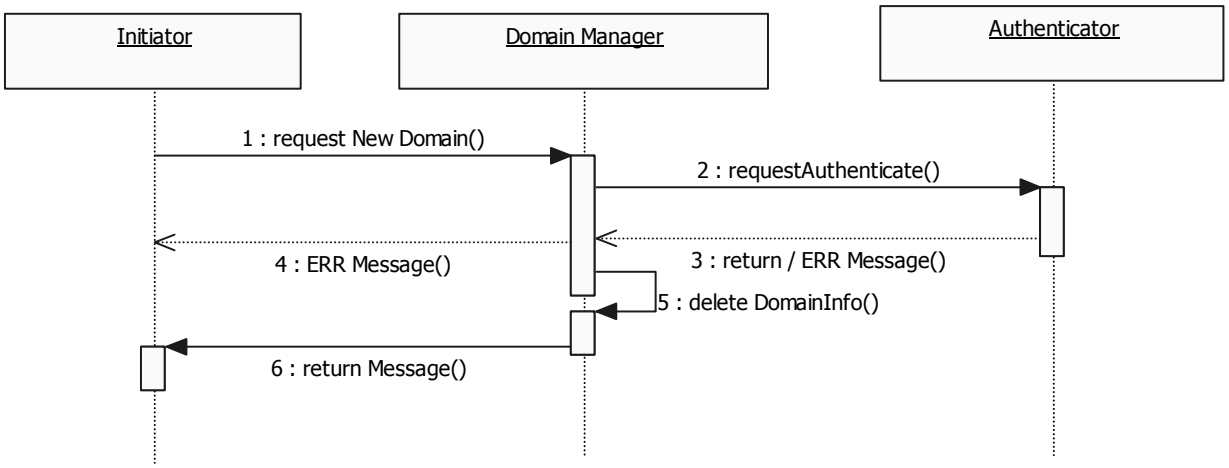


Figure 171 — Revoke Domain

- `requestRevokeDomain`

Parameters	`DomainInfo domainInfo`
Semantics	This function requests revoking specific domain requested.

- `deleteDomainInfo`

Parameters	`DomainInfo domainInfo`
Semantics	This function removes the domain associated with specific domain information. It returns `true` if there is no error occurred, otherwise, `false` is returned.

9.1.3.1.5.3 Update Domain

Domain information such as domain key needs to be updated with new key occasionally. To update a domain, a user device requests updating domain to DM. When the DM receives Domain update message from a user device, the message is transferred to Authenticator for verification. If the information is authenticated, DM updates Domain and returns the result to an initiator.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

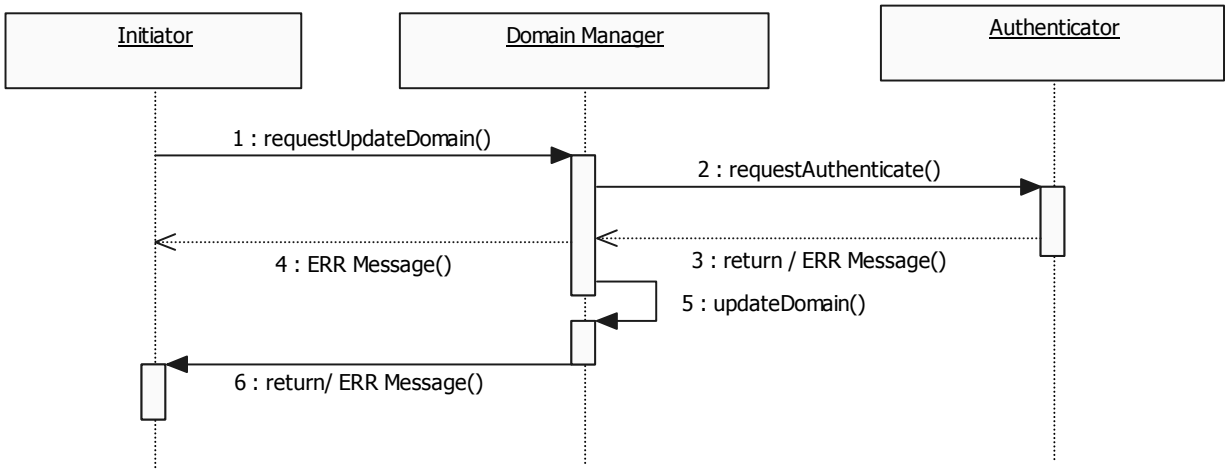


Figure 172 — Update Domain

- `requestupdateDomain`

Parameters	`DomainInfo domainInfo`
	
	`DomainInfo newDomainInfo`

Semantics	This function is to request update of the domain information associated with specific domain. It returns `true` if there is no error occurred, otherwise, `false` is returned.

- `updateDomain`

Parameters	`DomainInfo domainInfo`
Semantics	This function updates the domain information associated with specific domain. It returns `true` if there is no error occurred, otherwise, `false` is returned.

9.1.3.1.5.4 Unsubscribe from Domain

A device needs to be unsubscribed from a domain, if the device is not available anymore. To unsubscribe from the domain, domain administrator requests removal of the device from the domain to DM. When the relevant information given by domain administrator is verified by Authenticator, the DM processes to remove the device from the domain.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

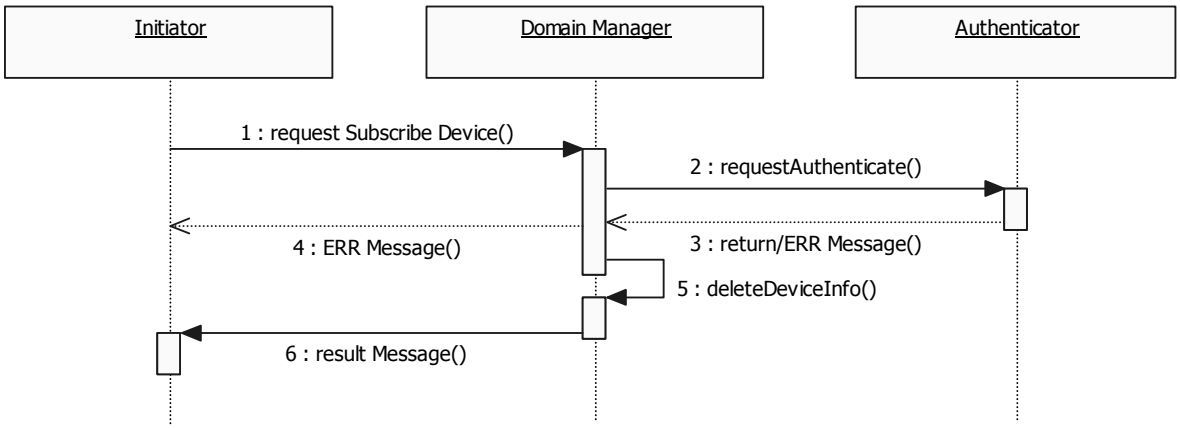


Figure 173 — Unsubscribe Device

- `requestUnsubscribeDevice`

Parameters	`DomainInfo domainInfo`
Semantics	This function requests removing a device indicated in domain information from the domain. It returns `true` if there is no error occurred, otherwise, `false` is returned.

- `deleteDeviceInfo`

Parameters	`DomainInfo domainInfo`
Semantics	This function unsubscribes a device specified in domain information from the Domain. It returns `true` if the device is unsubscribed successfully, otherwise, `false` is returned.

9.1.3.2 Tool Interfaces

Figure 174 — Example of IPMP processing shows an example of an IPMP procedure. the IPMP information of MPEG-2/4 IPMPX is extracted and then IPMP tool is executed. Specifically, according to the IPMP information, proper IPMP tools are retrieved and then executed. If there is no proper tool, a tool manager should get missing tools from remote URI in the IPMP information. During the processing of MPEG-2/4, whenever and wherever IPMP processing is required, each tool performs its own IPMP functions.

```
typedef ByteArray IPMPtoolInstance;
typedef ByteArray IPMPDescriptor; // Data type for IPMP Descriptor
typedef ByteArray IPMPControlInfoDescriptor;
typedef ByteArray IPMPData;
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

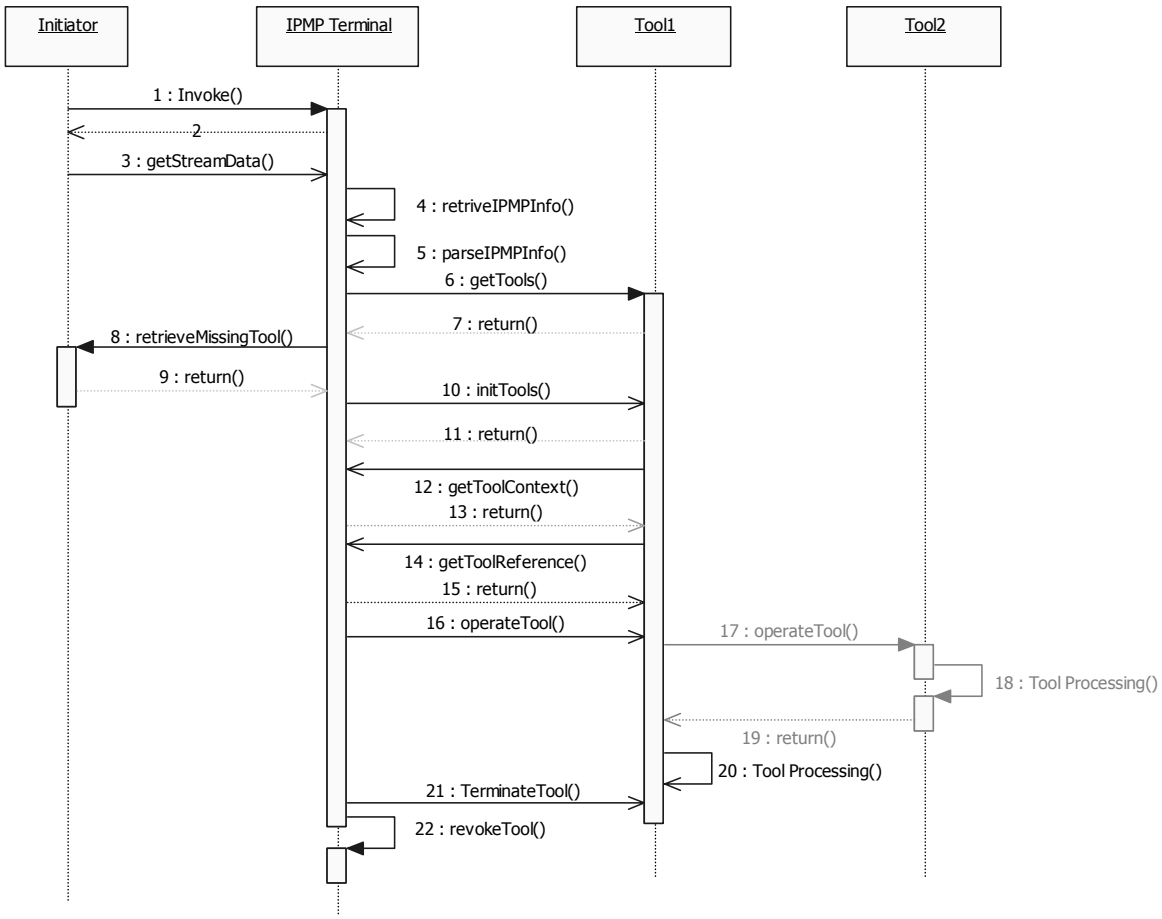


Figure 174 — Example of IPMP processing

9.1.3.2.1 General Tool Processing Operation

In IPMP architecture, IPMP conditions in IPMP information specify which IPMP processing is required. From these conditions, the required IPMP tools are selected and initialized. Then, as the multimedia data are processed, the tools do their duties based on IPMP messages. Figure 175 — IPMP Tool processing shows a procedure of general IPMP tool processing.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

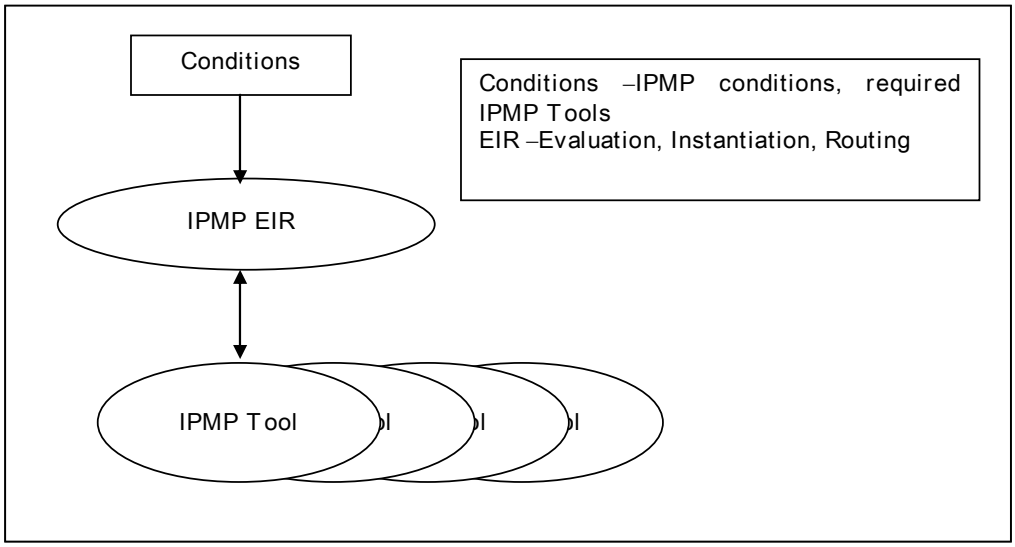


Figure 175 — IPMP Tool processing

In the followings IPMP functions required during general tool operations are represented. Note that these functions support all kind of IPMP tool operations (e.g., encryption, decryption, and watermarking etc.)

9.1.3.2.1.1 Access IPMP Tool Identifiers

The IPMP Tool Interface functions allow information to be exchanged between the IPMP Tool Terminal and Tool requiring information to identify the IPMP Tool or IPMP Tool Group. Figure 174 — Example of IPMP processing illustrates one example of using identifiers. In this figure, the processing in the ellipse is proposed for a tool (Tool1) to get required information to interact with another tool (Tool2).

`MmIIPMPToolIdentifier` gives information such as Tool Conext ID and Tool Group Refernece regarding a Tool or IPMP Tool Group which IPMP Terminal is processing currently.

```
interface MmIIPMPToolIdentifier {
UUID getToolContext(in IPMPDescriptor ipmpInfo);
UUID getToolGroupReference(in UUID toolID);
UUID getToolReference(in UUID toolID);
}
```

- `getToolContext`

Parameters	`UUID toolID`
Semantics	This function retrieves unique Tool Context ID for the tool with specific toolID.

- `getToolGroupReference`

Parameters	`UUID toolID`
Semantics	This function retrieves the Reference of the associated IPMP Tool Group with specific toolID.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- `getToolReference`

Parameters	`UUID toolID`
Semantics	This function retrieves the Reference of the specific single IPMP Tool which is not in the IPMP Tool Group with given toolID.

9.1.3.2.1.2 Tool connection

For connection of a tool and terminal, the terminal module can request for a tool its specific parametric description. This function allows a terminal to query a tool as for support for a specific parametric description.

```
interface MmIIPMPTerminal extends MmIIPMPToolIdentifier
{ 3b0b850d-69f3-4e6d-a1d4-4f3874634750 }
{
    Bool retriveIPMPInfo(in Sequence sequence);
    Bool retriveMissingTool(in URI remoteToolURI, in UUID toolID);
    Bool parseIPMPInfo(in IPMPControlInfoDescriptor ipmpInfo);
    Bool getTools(in UUID toolID, in IPMPtoolInstance ti);
    Bool revokeTool(in UUID toolID);
    Void terminateTool(in UUID toolID);
}


/**
*  This is interface for operating to be defined by MPEG IPMP.
*
**/
interface MmIIPMPTool { 0943d06d-5838-44a3-8b77-96ae49a99596 } {
    Bool operateTool(in IPMPDescriptor ipmpInfo,
    inout Stream stream, in IPMPData ipmpData);
    Void toolProcessing(in UUID toolID);
    Void initTool();
}
```

- queryToolParametricCapability

Parameters	in ByteArray toolParamDesc
Semantics	This function is called for query of a tool as for support for a specific parametric description.

- getStreamData

Parameters	Void
Semantics	This function requests multimedia sequence and gets it into an IPMP terminal.

- `retrieveIPMPInfo`

Parameters	Sequence sequence
Semantics	This function retrieves IPMP info from multimedia sequence. if IPMP info exists, it returns IPMP info as an `IPMPInfo` desriptor

- `parseIPMPInfo`

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters	`IPMPControlInfoDescriptor` ipmpInfo
Semantics	This function parses IPMP info and extracts tool lists required.

- `getTools`

Parameters	UUID toolID
Semantics	This function gets a proper tool stored in an IPMP device. If the tool exists in the device, it returns the tool instance, otherwise it returns error messages or call `retrieveMissingTool` to find the missing tool.

- `retrieveMissingTool`

Parameters	URI remoteToolURI UUID toolID
Semantics	This function retrieves missing tools from remote source remoteToolURI. If the tools are obtained, it returns TRUE or SUCCESS; otherwise it returns FAIL or FALSE.

- `initTool`

Parameters	UUID toolID
Semantics	This function connects a specified tool with an IPMP terminal and initializes the tool.

- `operateTool`

Parameters	`IPMPDescriptor` Stream stream IPMPData ipmpData
Semantics	This function executes a specified tool for the given stream data using IPMP info and ipmpData.

- `terminateTool`

Parameters	UUID toolID
Semantics	This function terminates a specified tool and disconnects the tool from IPMP terminal.

- `revokeTool`

Parameters	UUID toolID
Semantics	This function revokes a specified tool.

9.1.3.2.1.3 User interaction

These interface functions allow information to be exchanged between the User and an entity requiring information from the User.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- userQuery

Parameters	in ByteArray userQueryInfo
Semantics	This function is used for user query

- userQueryResponse

Parameters	in ByteArray userQueryResponse
Semantics	This function is for a response for *userQuery*

9.1.3.2.2 IPMP Tool Functions

In MPEG IPMP, there is no limitation about IPMP tool functions. Therefore, any kind of functions can be inserted in an IPMP device if the functions are related to IPMP. Here, we present general functionalities used in DRM or IPMP services, which can be extended to support other IPMP related functions.

9.1.3.2.2.1 Authentication

Before authentication, the logical instance of the Tool with which authentication is to be performed should be initialized.

```
/**
*  This interface is for authentication tool.
*/

interface MmIAuthenticateTool extends MmIIPMPTool
             { b3024303-d6ae-44be-9fe0-fbb0657c826f } {
       Bool initAuthentication(in ByteArray initAuthData);
       Bool authenticate(in Stream *stream,in IPMPData ipmpData);
};
```

- initAuthentication

Parameters	in ByteArray initAuthData
Semantics	This function initializes a logical instance of the authentication tool.

- `Authenticate`

Parameters	Stream (sequence pointer) stream `IPMPDescriptor` ipmpInfo IPMPData ipmpData (optional)
Semantics	This function authenticates a device or a user by an MPEG stream according to IPMP information.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.1.3.2.2.2 Encryption/Decryption

For using IPMP encryption tool, following functions shall be called on M3W:

```
/**
*  This interface is for encryption/decryption tool.
*/
typedef sequence <String, 1024> List;
typedef sequence <Buffer, 1024> KeyList;


interface MmIEncryptionTool extends MmIIPMPTool
      { 9d8da62a-180f-4051-8a73-8313b25f5a0b } {
       List queryEncryptionAlgorithms();
       Bool encrypt(inout Stream *stream, in IPMPData ipmpData);
    Bool decrypt(inout Stream *stream, in IPMPData ipmpData);
       Int32 initEncryption(in String algorithm, in Buffer params);
       Int32 updateEncryptionKeys(in KeyList keys);
       Int32 stopEncryption(Void);
       Int32 initDecryption(in String algorithm, in Buffer params);
       Int32 updateDecryptionKeys(in KeyList keys);
       Int32 stopDecryption(Void);
};
```

- queryEncryptionAlgorithms

Parameters	out stringArray AlgorithmList
Semantics	This function is called when a Registration Authority issue standard identifiers for established algorithms.

- initEncryption

Parameters	In Algorithm, PARAM params
Semantics	This method initializes content encryption tool. The return variable is a reference to the encrypted content or (if negative) an error code.

- updateEncryptionKeys

Parameters	In KeyList keys
Semantics	This method starts or continues content encryption using new or modified keys. An array of keys is used to allow for key changes.

- stopEncryption

Parameters	void
Semantics	This method stops content encryption

- initDecryption

Parameters	In Algorithm, PARAM params
Semantics	This method initializes content decryption. The return variable is a reference to the decrypted content or (if negative) an error code.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- updateDecryptionKeys

Parameters	in KeyList keys
Semantics	This method starts or continues content decryption using new or modified keys. An array of keys is used to allow for key changes.

- stopDecryption

Parameters	void
Semantics	This method stops content decryption

- Encrypt

Parameters	Stream (sequence pointer) stream `IPMPDescriptor` ipmpInfo IPMPData ipmpData (optional)
Semantics	This function encrypts an MPEG stream by a method indicated in IPMP information. The key information required in encryption can be given through ipmpData.

- Decrypt

Parameters	Stream (sequence pointer) stream `IPMPDescriptor` ipmpInfo IPMPData ipmpData (optional)
Semantics	This function decrypts encrypted an MPEG stream by a method indicated in IPMP information. The key information required in decryption can be given through ipmpData.

9.1.3.2.2.3 Watermarking

```
/**
*  This interface is for watermarking tool.
*/

interface MmIWatermarkTool extends MmIIPMPTool
       { e5caff8a-ea53-4778-ab7a-3c3d15cab839} {
       List queryWatermarkAlgorithms();
       Int32 stopWatermarkExtraction(Void);
       Int32 stopWatermarkInsertion(Void);
       Bool embedWatermark(inout Stream *stream, in IPMPData ipmpData);
       Bool detectWatermark(inout Stream *stream, in IPMPData ipmpData);
};
```

- initWatermarking

Parameters	in ByteArray initWatermarkingData
Semantics	This function is called for initializing a watermarking tool.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- queryWatermarkingAlgorithms

Parameters	out stringArray AlgorithmList
Semantics	This function is called when a Registration Authority issue standard identifiers for established algorithms.

- stopWatermarkInsertion

Parameters	Void
Semantics	This method stops watermark Insertion from a content stream.

- stopWatermarkExtraction

Parameters	Void
Semantics	This method stops watermark extraction from a content

- `embedWatermark`

Parameters	Stream stream `IPMPDescriptor` ipmpInfo IPMPData ipmpData (optional)
Semantics	This function embeds information as a watermark into an MPEG stream by a method indicated in IPMP information. If key information is required, it can be given through ipmpData.

- `detectWatermark`

Parameters	Stream stream `IPMPDescriptor` ipmpInfo IPMPData ipmpData (optional)
Semantics	This function detects watermarked information in an MPEG stream by a method indicated in IPMP information. If key information is required, it can be given through ipmpData.

9.1.3.2.3 Tool Update operation

IPMP system should support to update IPMP tool, for upgrading or updating the functionality of the tool. First, an IPMP terminal checks tool conditions, and if the condition says to update, the request is sent to remote source indicated in IPMP information. Finally, the requester gets the remote tools and update old tools.

```
/**
*  This interface is for updating Tool.
*/
interface MmIIPMPToolupdate { e97620ed-4459-4c72-b8c0-26b6232b53ca} {
       Bool validateToolCondition (in ToolInfoDescriptor toolInfo);
       Bool requestUpdate(in URI remoteToolURI, in UUID ToolID);
       Bool updateTool(in URI remoteToolURI, in UUID ToolID);
}
```


© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- `validateToolCondition`

Parameters	ToolInfoDescriptor toolInfo
Semantics	This function verifies tool conditions provided in Tool Info and checks whether the conditions are valid. If they are valid, it returns TRUE or SUCCESS; otherwise, it returns FAIL or FALSE.

- `requestUpdate`

Parameters	URI remoteToolURI UUID ToolID
Semantics	This function requests the updating of a specified tool in target URI

- `updateTool`

Parameters	URI remoteToolURI UUID ToolID
Semantics	This function updates the given tool from remote source, remoteToolURI. If the tool is updated, it returns TRUE or SUCCESS; otherwise, it returns FAIL or FALSE.

9.1.3.2.4 Tool Communications

Communications between different tools or IPMP devices should be supported by IPMP Interfaces. Following interfaces are basic interfaces for communications between each IPMP devices.

```
/**
*  This interface is for communicate between a both of tools.
*/

interface MmIIPMPToolCommunication { ebe56ab5-e465-4e72-b18d-e5b61a77c2b9} {
    Bool sendMessage(in UUID targetID, in Message message);
    Bool receiveMessage(in UUID sourceID, in Message message);
}
```

- `sendMessage`

Parameters	UUID targetID Message message
Semantics	This function is used for sending messages

- `recieveMessages`

Parameters	UUID sourceID Message message
Semantics	This function is used for receiving messages

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2 Governance interface suites

9.2.1 ISecurityManager

9.2.1.1 Concepts

The function of Security Manager is to provide secure interaction between M3W Service and external entities. This may include the interaction to acquire Service through download mechanism and invoke Service from remote peer. Security Manager also provides interface for governance/permission checking for accessing M3W Service and/or platform's resources.

9.2.1.2 Types and Constants

9.2.1.2.1 Public Types & Constants

9.2.1.2.1.1 Error Codes

Signature

```
const rcResult RC_ERR_SecMan_NO_SUCH_PARAMETER = 0x00001101
const rcResult RC_ERR_SecMan_CANNOT_ESTABLISH = 0x00001102
const rcResult RC_ERR_SecMan_UNKNOWN_SESSION = 0x00001103
const rcResult RC_ERR_SecMan_CANNOT_VERIFY = 0x00001104
```

Qualifiers

— Error-codes

Description

The non-standard error codes that can be returned by functions of this logical component

Constants

Name	Description
RC_ERR_SecMan_NO_SUCH_PARAMETER	This error is returned when a particular function is invoke with unknown or unrecognized parameter
RC_ERR_SecMan_CANNOT_ESTABLISH	This error is returned when a request to setup a secure channel is failed to fulfilled
RC_ERR_SecMan_UNKNOWN_SESSION	This error is returned when a request to close a session with unknown session object

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
RC_ERR_SecMan_CANNOT_VERIFY	This error is returned when an attemtp to verify an access (to a Service or platform's resource cannot be completed

9.2.1.2.1.2 srcSecMan_PeerLocation

Signature

```
struct _rcSecMan_PeerLocation {
    String location;
} rcSecMan_PeerLocation, *prcSecMan_PeerLocation;
```

Qualifiers

— struct-element

Description

Data structure used to pass the address of a peer location

9.2.1.3 Logical Component

9.2.1.3.1 Interface-RoleModel

Figure 176 — Security Manager's Interface-Role Model, depicts the interface-role model of the Security Manager (grey box). It shows the interface and the roles involved with this component.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

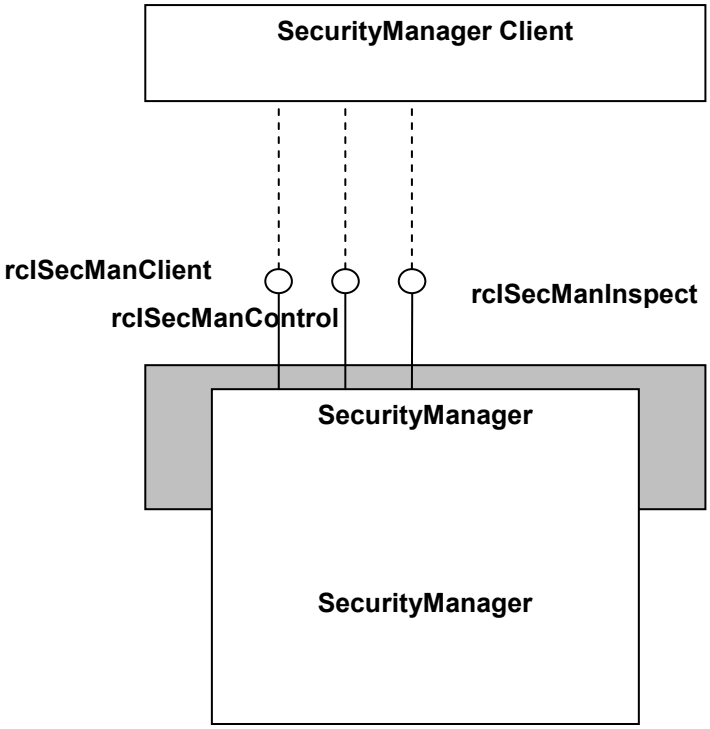


Figure 176 — Security Manager's Interface-Role Model

A Security Manager Logical Component contains the Security Manager Role that provides the rclSecManClient interface. This interface can be used by a client to request secure communication operations (i.e. for secure download). Security Manager Role has an optional interface to control the knowledge/state of the Service Manager - rclSecManControl. This involves registering and un-registering the parameter of the security state of the middleware. The form of the parameter depends on the implementation of this Logical Component. One example of implementing the parameter can be through metadata for security as described in Annex B of the ISO/IEC 23004-5 (Component download).

9.2.1.3.2 Diversity

9.2.1.3.2.1 Provided Interfaces

Role	Interface	Presence
Security Manager	`rcISecManClient`	`Mandatory`
Security Manager	rclSecManControl	Optional
Security Manager	rclSecManInspect	Optional

9.2.1.3.2.2 Configurable Items

None

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.1.3.2.3 Constraint

None

9.2.1.3.3 Instantiation

9.2.1.3.3.1 Object Created

Security Manager logical component and role are always optional. But if it is implemented, it's multiplicity shall be 1 in the middleware

Type	Object	Multiplicity
SecurityManager	`SecurityManager`	1

9.2.1.3.3.2 Object Created

The following constraints apply to the initial state of a logical component instance:

--- None

9.2.1.3.4 Execution Constraints

If implemented with thread, it shall be thread-safe

9.2.1.4 Roles

9.2.1.4.1 Security Manager

Signature

```
role SecurityManager {}
```

Qualifiers

— TBD

Description

Security Manager provides the rclSecManClient interface. This interface can be used by a client to request secure communication operations (i.e. for secure download).

Instantiation

Security Manager is optionally instantiated.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.1.5 Interfaces

9.2.1.5.1 rclSecManClient

Qualifiers

None

Description

This interface of Security Manager Role provides facilities for execute secure communications.

Interface ID

`584a62d7-d832-44e4-97ca-81cfa3b84171`

Remarks

None

9.2.1.5.1.1 OpenSecureSession

Signature

```
rcResult OpenSecureSession(
    [in] pUUID peerId,
    [in] rcSecMan_PeerLocation location,
    [out] object session
);
```

Qualifiers

— synchronous

— thread-safe

Description

Create/open a secure session/channel for communication. The security manager will contact the specified peer and negotiate (the agreed security parameter) to open a secure communication channel.

Parameters

Name	Description
peerId	Reference to the UUID of the entity that the Security Manager shall contact. The entity could be the Security Manager of the peer.
`location`	The location of the peer.
`session`	`An object of a secure channel. Depending on the implementation`, it can be a socket or other means of communication object.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values

Only non-standard error values are listed

Name	Description
RC_ERR_SecMan_CANNOT_ESTABLISH	The requested session cannot be established.

Precondition

— True

Action

Open a secure session/channel.

Post condition

— A secure session/channel for communication is available

9.2.1.5.1.2 CloseSecureSession

Signature

```
rcResult OpenSecureSession(
    [in] object session,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Close a secure session/channel for communication. The security manager will contact the specified peer and negotiate (the agreed security parameter) to close a secure communication channel.

Parameters

Name	Description
session	An object of a secure channel. Depending on the implementation, it can be a socket or other means of communication object.
retValue	A boolean value. True -> successfully close the session; False -> otherwise.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values

Only non-standard error values are listed

Name	Description
RC_ERR_SecMan_UNKNOWN_SESSION	The given session is unknown/not exist.

Precondition

— True

Action

Close a secure session/channel. Notify the peer to close the session.

Post condition

— The given session is closed and removed from system memory

9.2.1.5.1.3 SetPreferredParameterValue

Signature

```
rcResult SetPreferredParameterValue(
    [in] object session,
    [in] String paramType,
    [out] String paramValue,
);
```

Qualifiers

— synchronous

— thread-safe

Description

Use for negotiating the security parameters. Security Manager Implementation shall contact the peer and Set the peer's preferred value for the parameter.

Parameters

Name	Description
session	An object of a secure channel. Depending on the implementation, it can be a socket or other means of communication object.
paramType	A string of parameter type
paramValue	A string of parameter value

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values

Only non-standard error values are listed

Name	Description
RC_ERR_SecMan_NO_SUCH_PARAMETER	The given paramType is unknown/not exist.

Precondition

— True

Action

Set the value to the paramType. Check whether the value is ok. If OK return true otherwise return false. In case of receiving false return, repeating this action till the value for the paramType is agreed.

Post condition

— The value for the parameter is agreed.

9.2.1.5.1.4 SetSupportedSecurityParameters

Signature

```
rcResult SetSupportedSecurityParameters(
    [in] object session,
    [in] object metadata,
);
```

Qualifiers

— synchronous

— thread-safe

Description

Use for negotiating the security parameters. Security Manager Implementation shall contact the peer and send all its preference for the session's security parameters.

Parameters

Name	Description
Session	An object of a secure channel. Depending on the implementation, it can be a socket or other means of communication object.
Metadata	Metadata expression that model the security parameters. An example of such metadata can be found in in Annex B, ISO/IEC 23004-5.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition

— True

Action

Set the value to the paramType. Check whether the value is ok. If OK return true otherwise return false. In case of receiving false return, repeating this action till the value for the paramType is agreed.

Post condition

— The value for the parameter is agreed.

9.2.1.5.1.5 GetPreferredParameterValue

Signature

```
rcResult SetPreferredParameterValue(
    [in] object session,
    [in] String paramType,
    [out] String paramValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Use for negotiating the security parameters. Security Manager Implementation shall contact the peer and get the preferred parameter value

Parameters

Name	Description
Session	An object of a secure channel. Depending on the implementation, it can be a socket or other means of communication object.
paramType	A string of parameter type
paramValue	A string of parameter value

Return Values

Only non-standard error values are listed

Name	Description
RC_ERR_SecMan_NO_SUCH_PARAMETER	The given paramType is unknown/not exist.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Precondition

— True

Action

Get the value to the paramType.

Post condition

— The value of the parameter is known

9.2.1.5.1.6 VerifyAccessToService

Signature

```
rcResult VerifyAccessToService(
    [in] String accessType,
    [in] prcIService *accessor,
    [in] pUUID srvID,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Verify whether an access request to a Service can be granted or not.

Parameters

Name	Description
accessType	A string of access type. It can be access such as execute, etc.
accesssor	Reference to the accessor. It is needed to know who is the principal of the access
srvID	The id of Service that the accessor wants to access.
retValue	A boolean value. True -> access is granted; False -> otherwise.

Return Values

Only non-standard error values are listed

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
RC_ERR_SecMan_CANNOT_VERIFY	Cannot perform the verification task.

Precondition

— True

Action

Check into the implementation's knowledge (can be a collection of access license). If there exist a license or grant that grants the access, return true, false otherwise.

Post condition

— none.

9.2.1.5.1.7 VerifyAccessToResource

Signature

```
rcResult VerifyAccessToResource(
    [in] String accessType,
    [in] prcIService *accessor,
    [in] String resourceType,
    [out] object accessKey
);
```

Qualifiers

— synchronous

— thread-safe

Description

Verify whether an access request to a Service can be granted or not.

Parameters

Name	Description
accessType	A string of access type. It can be access such as read, write, open (socket), etc.
accesssor	Reference to the accessor. It is needed to know who is the principal of the access
resourceType	The type of resource that the accessor wants to access.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
accessKey	accessKey is an object that can be used by the accessor then reequesting the actual resource for the resource pool/manager. It is used for matching whether the access has really been granted by Security Manager or not. A null accessKey tells that access is not granted.

Return Values

Only non-standard error values are listed

Name	Description
RC_ERR_SecMan_CANNOT_VERIFY	Cannot perform the verification task.

Precondition

— True

Action

Check into the implementation's knowledge (can be a collection of access license). If there exist a license or grant that grants the access, return true, false otherwise.

Post condition

— none.

9.2.1.5.2 rclSecManControl

Qualifiers

None

Description

This interface of Security Manager Role provides facilities for adding and removing the security parameter.

Interface ID

`4d2073a7-1f35-4bf2-89d1-96c677be8628`

Remarks

None

9.2.1.5.2.1 AddSecurityParameter

Signature

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
rcResult AddSecurityParameter(
    [in] prcIService *invoker,
    [in] String paramType,
    [in] String paramValue,
    [in] Int32 preference,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Insert a value of a parameter to the Security Manager's knowledge

Parameters

Name	Description
invoker	The entity that invoke this API. It is needed to verify whether the invoker has rights to add the parameter's value or not
paramType	A string of parameter type
paramValue	A string of parameter value
preference	Show the preference value of that paremeter. A parameter may have more that one value. Each value has its preference to show which one is more likely to be used. The higher the preference value the more likely it is used.
retValue	A boolean value. True -> value has been added; False -> otherwise.

Precondition

— True

Action

Add a parameter's value.

Post condition

— A value is added in the Security Manager's parameters

9.2.1.5.2.2 RemoveSecurityParameter

Signature

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
rcResult AddSecurityParameter(
    [in] prcIService *invoker,
    [in] String paramType,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Remove a value of a parameter from the Security Manager's knowledge

Parameters

Name	Description
invoker	The entity that invoke this API. It is needed to verify whether the invoker has rights to remove the parameter's value or not
paramType	A string of parameter type
retValue	A boolean value. True -> value has been removed; False -> otherwise.

Return Values

Only non-standard error values are listed

Name	Description
RC_ERR_SecMan_NO_SUCH_PARAMETER	The given paramType is unknown/not exist.

Precondition

— True

Action

Remove a parameter's value.

Post condition

— A value is removed from the Security Manager's parameters

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.1.5.3 rclSecManInspect

Qualifiers

None

Description

This interface of Security Manager Role provides facilities for checking what kind of security parameters used by the Service Manager's implementation.

Interface ID

`d75702d4-3adc-49c4-bf99-fda632191493`

Remarks

None

9.2.1.5.3.1 GetSecurityProfile

Signature

```
rcResult AddSecurityParameter(
    [out] object securityProfile
);
```

Qualifiers

— synchronous

— thread-safe

Description

Get the information about security profile (parameters) used by the Security Manager's implementation

Parameters

Name	Description
securityProfile	Security Manager's security profile. It can be expressed in metadata form. An example of such metadata can be found in Annex B of the ISO/IEC 23004-5 (Component download).

Precondition

— True

Action

Return the knowledge pool.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Post condition

— none

9.2.1.6 Usage Example (Informative)

9.2.1.6.1 Example 1: Governing Access to Service & Resource

Access to the Services in the middleware and the platform resources must be governed so that they are only granted to the authorized entities only. For ensuring the governance, prior to any access attempt, the entity that tries to access needs to be checked about its access rights by the Security Manager. When a user application wants to access a Service:

☐ User application send a access request to the Service Manager;

☐ The Service Manager gets the Security Manager to check whether that user application has a rights to use the Service or not;

☐ The Security Manager checks its rights collection to see whether there is a match & valid rights that grants access to the requesting service by the user application under the fully satisfying conditions;

☐ The Security Manager reports the rights validation result to the Service Manager;

☐ If the rights validation result from the Security Manager turns out to to positive, then the Service Manager can proceed to instantiate and give over the handler of the Service to the user application; and

☐ Otherwise, the Service Manager refuses to proceed.

In other case, when a Service needs to access the platform's resources, it must be ensured that the middleware is fully aware of this access and be sure that the access would not cause any harm to the system/terminal or other applications that run simultaneously on top of the middleware with its clients. When a Service wants to access any platform's resource:

☐ The Service sends a request to Security Manager to get permission to access the platform's resource;

☐ Security Manager checks its platform's resource access policy;

☐ If there is access rights for the request, Security Manager creates an access ID (random number);

☐ Security Manager sends that access ID to RRE and the requesting Service;

☐ Service asks RRE for the system property (all the platform resources are considered as system properties) and gives the access ID;

☐ RRE matches the access IDs it receives from Security Manager and the Service, it also matches the Service ID; and

☐ If everything is match, the access will be allowed and the Service may continue its process.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In order for the Security Manager to be able to verify the access rights to the Service or to the platform's resources, it needs to be equipped with a list of access rights. This access rights contains information such as:

- ☐ Type of permission (access to Service, access to file system, access to memory, etc);
- ☐ Subject. Entity to which the rights is granted;
- ☐ Object. Entity that will be accesses;
- ☐ Type of access (execute, read, write, etc); and
- ☐ Condition of access (if necessary).

Figure 177 — Access from application to middleware's service, shows the sequence in the case of access from a user application to the middleware's service.

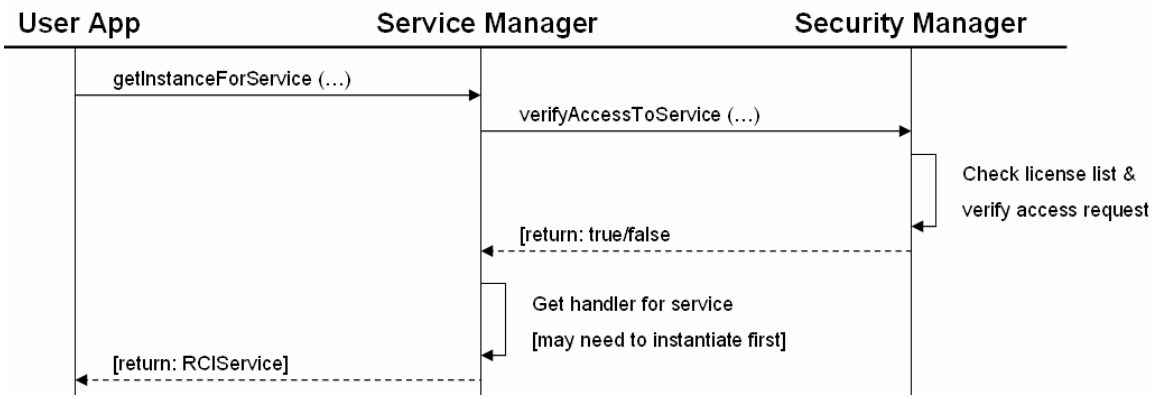


Figure 177 — Access from application to middleware's service

Figure 178 — Access from middleware's service to platform's resources, shows the sequence diagram in the case of access from middleware's service to the platform's resources

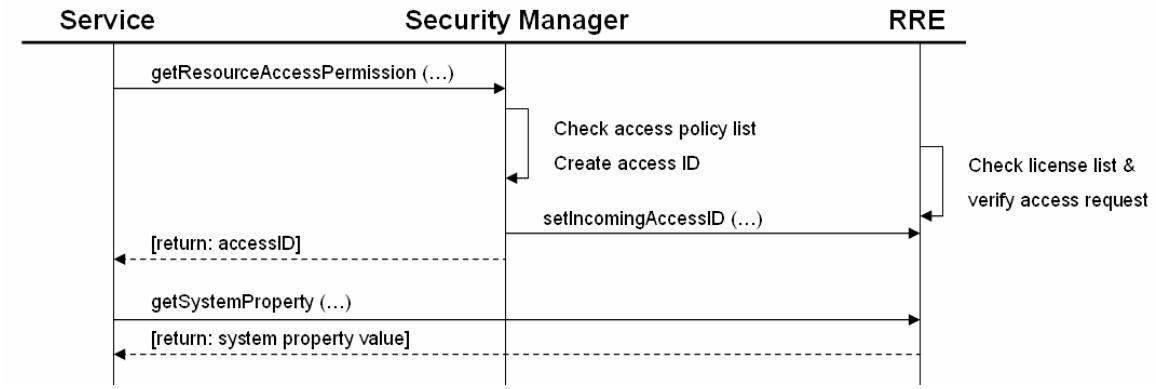


Figure 178 — Access from middleware's service to platform's resources

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A
(normative)

Global types

A.1 Introduction

This Annex defines the non-basic data types and constants that are “shared” between two or more interface suite specifications. It does not define any interfaces.

A.2 Concepts

A.2.1 Global Types

A global type is a data type that is used in two or more logical components and that is not a basic type as defined in ISO/IEC 23004-1 (Annex on basics types). A typical example of a global type is uhRect_t defining rectangles in an x-y coordinate system.

Besides definitions of global data types this document may also contain definitions of global constants. The arguments that apply to global data types also apply to global constants.

A.3 Types & Constants

A.3.1 Public Types & Constants

A.3.1.1 uhXY_t

Signature

```
typedef struct _uhXY_t
{
    Int32 x;
    Int32 y;
} uhXY_t, *puhXY_t;
```

Qualifiers

None.

Description

Defines values that represent coordinates in an x-y coordinate system.

Fields

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
x	x-coordinate.
y	y-coordinate.

A.3.1.2 uhRect_t

Signature

```
typedef struct _uhRect_t
{
    uhXY_t ul;
    uhXY_t lr;
} uhRect_t, *puhRect_t;
```

Qualifiers

None.

Description

Defines values that represent rectangles in an x-y coordinate system (with pixels as units). A rectangle is defined by its upper-left and lower-right corners. The upper-left corner is part of the rectangle; the lower-right is just outside the rectangle. For example, a square with width 100, height 100, and starting at (10,10) has an upper-left coordinate of (10,10) and a lower-right coordinate of (110,110).

Fields

Name	Description
ul	Upper-left coordinate of the rectangle (is part of the rectangle).
lr	Lower-right coordinate of the rectangle (*not* part of the rectangle).

A.3.1.3 uhPixFmtCls_t

Signature

```
typedef enum _uhPixFmtCls_t
{
    uhPixFmtCls_Yuv  = 0x00000010,
    uhPixFmtCls_Rgb  = 0x00000020,
    uhPixFmtCls_Clut = 0x00000008
} uhPixFmtCls_t, *puhPixFmtCls_t;
```

Qualifiers

enum-element

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description

Defines values representing the possible pixel format classes for video. The pixel format classes are defined by the types uhPixYuvFmt_t, uhPixRgbFmt_t and uhPixClutFmt_t, each defining a number of “pixel format types”.

Values

Name	Description
uhPixFmtCls_Yuv	Represents the YUV pixel format class defined by the type uhPixYuvFmt_t.
uhPixFmtCls_Rgb	Represents the RGB pixel format class defined by the type uhPixRgbFmt_t.
uhPixFmtCls_Clut	Represents the CLUT pixel format class defined by the type uhPixClutFmt_t.

Remarks

The enumerated values have been chosen so as to be binary compatible with existing video format types.

A.3.1.4 uhPixFmtClsSet_t

Signature

```
typedef Int32 uhPixFmtClsSet_t, *puhPixFmtClsSet_t;
```

Qualifiers

enum-set

Description

Values of this type represent sets of values of type uhPixFmtCls_t.

A.3.1.5 uhPixFmtType_t

Signature

```
typedef UInt32 uhPixFmtType_t, *puhPixFmtType_t;
```

Qualifiers

sub-type

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description

Defines the “union” of the pixel format classes for video as a subrange of the integers. That is, a value of this type is a value of type uhPixYuvFmt_t, uhPixRgbFmt_t or uhPixClutFmt_t. In order to interpret a value of this type correctly the pixel format class should be indicated. For example, the value 0x00000004 can be interpreted in one of three ways:

— as the value uhPixYuvFmt_Vyuy of pixel format class uhPixYuvFmt_t;

— as the value uhPixRgbFmt_Argb1555 of pixel format class uhPixRgbFmt_t;

— as the value uhPixClutFmt_RgbClut4bpp of pixel format class uhPixClutFmt_t.

Constraints

Each value of type uhPixFmtType_t is a valid value of at least one of the following types: uhPixYuvFmt_t, uhPixRgbFmt_t or uhPixClutFmt_t.

Remarks

A value of this type is normally used in combination with a value of type uhPixFmtCls_t to indicate the pixel format class the value belongs to.

A.3.1.6 uhPixFmtTypeSet_t

Signature

```
typedef Int32 uhPixFmtTypeSet_t, *puhPixFmtTypeSet_t;
```

Qualifiers

enum-set

Description

Values of this type represent sets of values of type uhPixFmtType_t.

A.3.1.7 uhPixYuvFmt_t

Signature

```
typedef enum _uhPixYuvFmt_t
{
    uhPixYuvFmt_Uyvy = 0x00000001,
    uhPixYuvFmt_Yuy2 = 0x00000002,
    uhPixYuvFmt_Vyuy = 0x00000004
} uhPixYuvFmt_t, *puhPixYuvFmt_t;
```

Qualifiers

enum-element

Description

Defines the possible format types of the YUV format class.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Name	Description
uhPixYuvFmt_Uyvy	Packed YUV422, bytes: U0,Y0,V0,Y1.
uhPixYuvFmt_Yuy2	Packed YUV422, bytes: Y0,U0,Y1,V0.
uhPixYuvFmt_Vyuy	Packed YUV422, bytes: V0,Y1,U0,Y0.

A.3.1.8 uhPixYuvFmtSet_t

Signature

```
typedef Int32 uhPixYuvFmtSet_t, *puhPixYuvFmtSet_t;
```

Qualifiers

enum-set

Description

Values of this type represent sets of values of type uhPixYuvFmt_t.

A.3.1.9 uhPixRgbFmt_t

Signature

```
typedef enum _uhPixRgbFmt_t
{
    uhPixRgbFmt_Xrgb1555 = 0x00000040,
    uhPixRgbFmt_Argb1555 = 0x00000004,
    uhPixRgbFmt_Xrgb4444 = 0x00000080,
    uhPixRgbFmt_Argb4444 = 0x00000100,
    uhPixRgbFmt_Rgb565   = 0x00000008,
    uhPixRgbFmt_Xrgb8888 = 0x00000010,
    uhPixRgbFmt_Argb8888 = 0x00000020
} uhPixRgbFmt_t, *puhPixRgbFmt_t;
```

Qualifiers

enum-element

Description

Defines the possible format types of the RGB format class. The x and a in the table below stand for don't care and alpha respectively.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Values

Name	Description
uhPixRgbFmt_Xrgb1555	u16: xrrr rrgg gggb bbbb
uhPixRgbFmt_Argb1555	u16: arrr rrgg gggb bbbb
uhPixRgbFmt_Xrgb4444	u16: xxxx rrrr gggg bbbb
uhPixRgbFmt_Argb4444	u16: aaaa rrrr gggg bbbb
uhPixRgbFmt_Rgb565	u16: rrrr rggg gggb bbbb
uhPixRgbFmt_Xrgb8888	u32: xxxxxxxx rrrrrrrr gggggggg bbbbbbbb
uhPixRgbFmt_Argb8888	u32: aaaaaaaa rrrrrrrr gggggggg bbbbbbbb

Remarks

The enumerated values have been chosen so as to be binary compatible with existing video format types.

A.3.1.10 uhPixRgbFmtSet_t

Signature

```
typedef Int32 uhPixRgbFmtSet_t, *puhPixRgbFmtSet_t;
```

Qualifiers

enum-set

Description

Values of this type represent sets of values of type uhPixRgbFmt_t.

A.3.1.11 uhPixClutFmt_t

Signature

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
typedef enum _uhPixClutFmt_t
{
    uhPixClutFmt_RgbClut1bpp  = 0x00000001,
    uhPixClutFmt_RgbClut2bpp  = 0x00000002,
    uhPixClutFmt_RgbClut4bpp  = 0x00000004,
    uhPixClutFmt_RgbClut8bpp  = 0x00000008,
    uhPixClutFmt_ArgbClut1bpp = 0x00000010,
    uhPixClutFmt_ArgbClut2bpp = 0x00000020,
    uhPixClutFmt_ArgbClut4bpp = 0x00000040,
    uhPixClutFmt_ArgbClut8bpp = 0x00000080,
    uhPixClutFmt_YuvClut1bpp  = 0x00000100,
    uhPixClutFmt_YuvClut2bpp  = 0x00000200,
    uhPixClutFmt_YuvClut4bpp  = 0x00000400,
    uhPixClutFmt_YuvClut8bpp  = 0x00000800,
    uhPixClutFmt_AyuvClut1bpp = 0x00001000,
    uhPixClutFmt_AyuvClut2bpp = 0x00002000,
    uhPixClutFmt_AyuvClut4bpp = 0x00004000,
    uhPixClutFmt_AyuvClut8bpp = 0x00008000
} uhPixClutFmt_t, *puhPixClutFmt_t;
```

Qualifiers

enum-element

Description

Defines the possible format types of the CLUT format class. The CLUT can contain RGB, αRGB, YUV, and αYUV entries. The x and a in the table below stand for don't care and alpha respectively.

Values

Name	Description
uhPixClutFmt_RgbClut1bpp	u32: xxxxxxxx rrrrrrrr gggggggg bbbbbbbb (2 CLUT entries)
uhPixClutFmt_RgbClut2bpp	u32: xxxxxxxx rrrrrrrr gggggggg bbbbbbbb (4 CLUT entries)
uhPixClutFmt_RgbClut4bpp	u32: xxxxxxxx rrrrrrrr gggggggg bbbbbbbb (16 CLUT entries)
uhPixClutFmt_RgbClut8bpp	u32: xxxxxxxx rrrrrrrr gggggggg bbbbbbbb (256 CLUT entries)
uhPixClutFmt_ArgbClut1bpp	u32: aaaaaaaa rrrrrrrr gggggggg bbbbbbbb (2 CLUT entries)
uhPixClutFmt_ArgbClut2bpp	u32: aaaaaaaa rrrrrrrr gggggggg bbbbbbbb (4 CLUT entries)
uhPixClutFmt_ArgbClut4bpp	u32: aaaaaaaa rrrrrrrr gggggggg bbbbbbbb (16 CLUT entries)

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description
uhPixClutFmt_ArgbClut8bpp	u32: aaaaaaaa rrrrrrrr gggggggg bbbbbbbb (256 CLUT entries)
uhPixClutFmt_YuvClut1bpp	u32: xxxxxxxx yyyyyyyy uuuuuuuu vvvvvvvv (2 CLUT entries)
uhPixClutFmt_YuvClut2bpp	u32: xxxxxxxx yyyyyyyy uuuuuuuu vvvvvvvv (4 CLUT entries)
uhPixClutFmt_YuvClut4bpp	u32: xxxxxxxx yyyyyyyy uuuuuuuu vvvvvvvv (16 CLUT entries)
uhPixClutFmt_YuvClut8bpp	u32: xxxxxxxx yyyyyyyy uuuuuuuu vvvvvvvv (256 CLUT entries)
uhPixClutFmt_AyuvClut1bpp	u32: aaaaaaaa yyyyyyyy uuuuuuuu vvvvvvvv (2 CLUT entries)
uhPixClutFmt_AyuvClut2bpp	u32: aaaaaaaa yyyyyyyy uuuuuuuu vvvvvvvv (4 CLUT entries)
uhPixClutFmt_AyuvClut4bpp	u32: aaaaaaaa yyyyyyyy uuuuuuuu vvvvvvvv (16 CLUT entries)
uhPixClutFmt_AyuvClut8bpp	u32: aaaaaaaa yyyyyyyy uuuuuuuu vvvvvvvv (256 CLUT entries)

A.3.1.12 uhPixClutFmtSet_t

Signature

```
typedef Int32 uhPixClutFmtSet_t, *puhPixClutFmtSet_t;
```

Qualifiers

enum-set

Description

Values of this type represent sets of values of type uhPixClutFmt_t.

A.3.1.13 uhColor_t

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Signature

```
typedef struct _uhColor_t
{
    UInt8 redOrY;
    UInt8 greenOrU;
    UInt8 blueOrV;
} uhColor_t, *puhColor_t;
```

Qualifiers

None.

Description

Defines values that represent 24 bit colors in the RGB or YUV color space.

Fields

Name	Description
redOrY	Red or Y component of the color.
greenOrU	Green or U component of the color.
blueOrV	Blue or V component of the color.

A.3.1.14 uhColorAlpha_t

Signature

```
typedef struct _uhColorAlpha_t
{
    UInt8 alpha;
    UInt8 redOrY;
    UInt8 greenOrU;
    UInt8 blueOrV;
} uhColorAlpha_t, *puhColorAlpha_t;
```

Qualifiers

None.

Description

Defines values that represent 32 bit colors in the ARGB or AYUV color space.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Fields

Name	Description
alpha	Transparency component of the color.
redOrY	Red or Y component of the color.
greenOrU	Green or U component of the color.
blueOrV	Blue or V component of the color.

A.3.1.15 uhTimeStamp_t

Signature

```
typedef struct _uhTimeStamp_t {
    UInt32 timeStampHigh;
    UInt32 timeStampLow;
} uhTimeStamp_t, *puhTimeStamp_t;
```

Qualifiers

None.

Description

Values of this type define time stamp structures. The resolution of the time stamp is 1 / 90000 of a second.

Fields

Name	Description
timeStampHigh	Bit 32 of the time stamp value.
timeStampLow	Bits 0 - 31 of the time stamp value.

A.3.1.16 UserInfo

Signature

```
Typedef struct _UserInfo {
URI AccessID
ByteArray AccessKey
} UserInfo, *UserInfo
```

Qualifiers

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description

Defines values that represent the information such as a unique access ID and the value of a key assigned to a user.

Fields

Name	Description
AccessID	Access user ID.
AccessKey	Key value.

A.3.1.17 DomainInfo

Signature

```
typedef struct _DomainInfo {
  URI DomainManagerID
  UserInfo AdminID
  UserInfo DomainID
  UserInfo DeviceID
} DomainInfo, *DomainInfo
```

Qualifiers

None.

Description

Defines values that represent the information of a domain.

Fields

Name	Description
AdminID	Information of domain administrator
DomainID	Information of domain
DeviceID	Information of device

A.3.2 Model Types & Constants

None.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.4 Logical Component

A.4.1 Interface-Role Model

Neither interfaces nor roles are defined in this annex. Hence the interface-role model as well as the “Roles” and “Interfaces” clauses are empty. These are therefore omitted.

A.4.2 Diversity

Since the M3W API is meant as a binary interface (like the interfaces in Microsoft COM) each global data type has a fixed representation. Hence there is no diversity in the global types.

A.4.3 Instantiation

No objects have to be created in order to instantiate this logical component. It can be viewed as a static logical component that is always available.

A.4.4 Execution Constraints

None.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
(informative)

Use Cases

B.1 Cancellation of an alarm (1)

The following figure depicts a scenario where a client sets an alarm and cancels it:

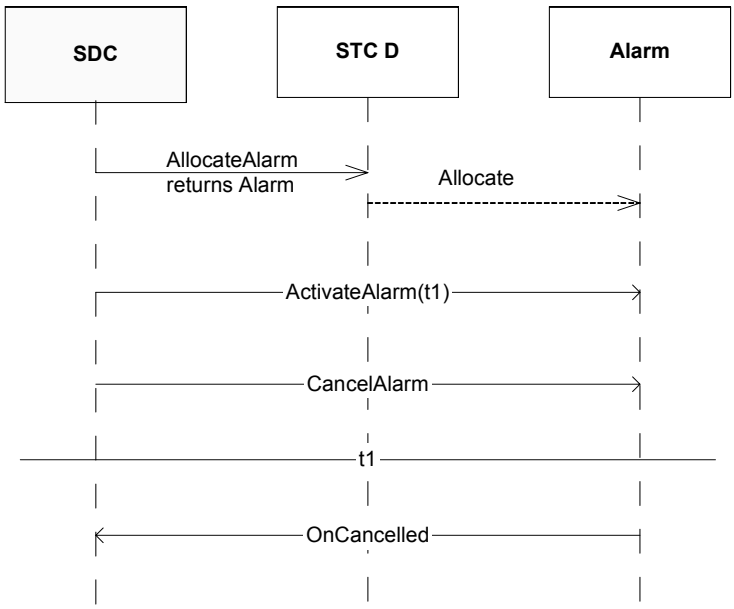


Figure B.1 — Cancellation of an alarm (1)

B.2 Cancellation of an alarm (2)

The following figure depicts a scenario slightly different from the previous one:

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

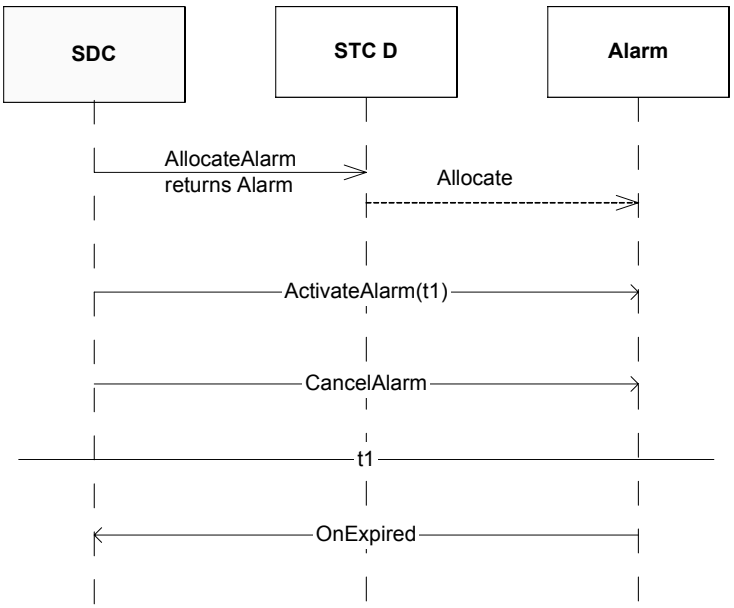


Figure B.2 — Cancellation of an alarm (2)

The client performs the same operations as in the first scenario, but because `CancelAlarm` is issued just before alarm should expire, the alarm expires before it is really cancelled by the STCD and the client is notified of the expiration instead of the cancellation, as was the case in the first scenario. The difference between these scenarios could be caused completely by the autonomous behavior of STCD and Alarm.

B.3 Erronous cancellation of an alarm

In the scenario depicted in the following figure, an alarm is cancelled that does not exist anymore (it expired). This results in no notifications, but the function returns UH_OK.

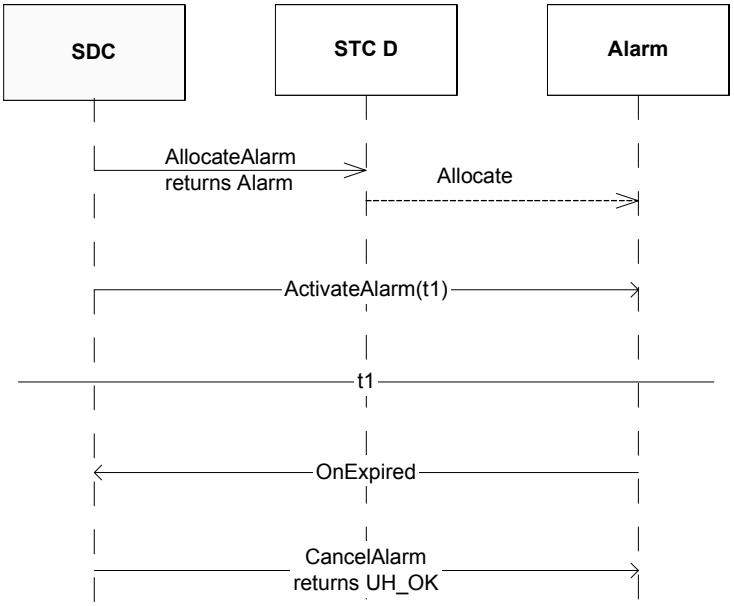


Figure B.3 — Erroneous cancellation of an alarm

NOTE The OnExpired and CancelAlarm calls could cross.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.4 Cancellation of an alarm on a discontinuity

In the scenario depicted in the following figure an alarm is cancelled because a discontinuity has made the alarm “unreliable“

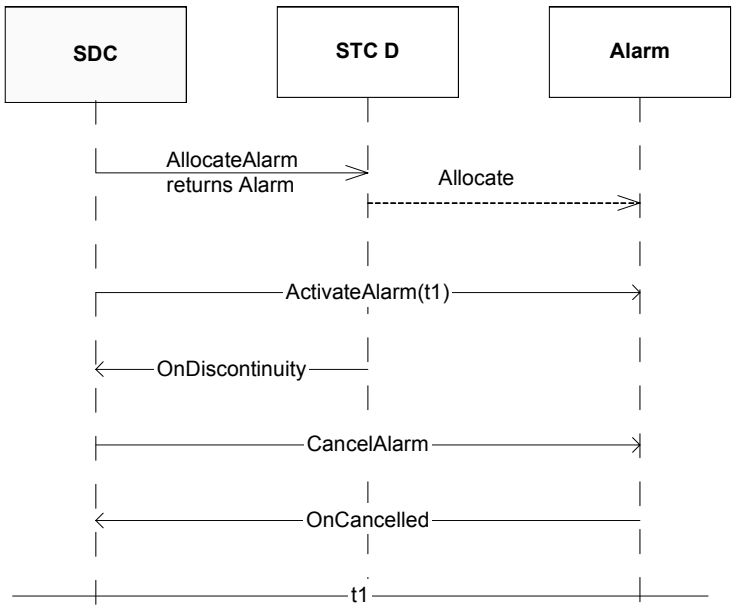


Figure B.4 — Cancellation of an alarm on a discontinuity

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex C
(informative)

Application notes

C.1 Using Reference Counting

In the literature about Microsoft COM the application and implementation of the `IUnknown` interface is well-documented; see e.g. [28]. Most of this information also applies to the `uhIUnknown` interface specfied in this document and will not be repeated here. A particularly subtle aspect of `(uh)IUnknown` is the reference counting mechanism supported by it which deserves some special attention. In order to make reference counting work correctly clients should follow a number of basic rules:

a) **Call AddRef before returning.** Functions that return interface pointers should always call AddRef on the pointer before returning. Therefore, you don't have to call `AddRef` after you get an interface pointer from a function.

b) **Call Release when you are done.** When you are finished with an interface pointer, you should call Release on that interface pointer.

c) **Call AddRef after assignment.** Whenever you assign an interface pointer to a pointer variable, call `AddRef`. Another way of saying this is that you should increment the reference count of an interface whenever you create another pointer to the interface.

In following the above rules care should be taken that a call of the Release function is performed on the same interface pointer as the corresponding call of the `AddRef` function. It is not safe to call `AddRef` on one interface of an object and the corresponding Release on another because this can lead to the reference count of an interface unintentionally becoming zero and the interface being "teared off" (deallocated) by the implementation. This cannot happen in implementations that use a reference count per object instead of per interface (the most common implementation), but clients cannot rely on reference counting being implemented this way. From the client's perspective, as implied by the specification of `uhIUnknown`, reference counting is always done on a per-interface basis.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex D
(informative)

Notational conventions

D.1 Dealing with uhIUnknown in API Specifications

The `uhIUnknown` interface is a special interface because of its omnipresence. Each role introduced in any M3W API specification is supposed to provide this interface and any interface is supposed to inherit from `uhIUnknown`. In order to make life easy a number of notational conventions are used in the other M3W API specifications that limit the specification overhead associated with `uhIUnknown`. Roughly speaking, these conventions amount to omitting `uhIUnknown`-related information from API specifications, as described below.

— Each role that provides/requires an interface also provides/requires the `uhIUnknown` interface. This is normally indicated by making the role specialize the Object or Client role defined in this document. Because this will unnecessarily complicate the diagrams without adding any real information, the convention is used that each role *R* that provides a M3W interface is implicitly assumed to specialize the Object role. Likewise, each role *S* that requires a M3W interface is implicitly assumed to specialize the Client role. This is schematically indicated in Figure D.1

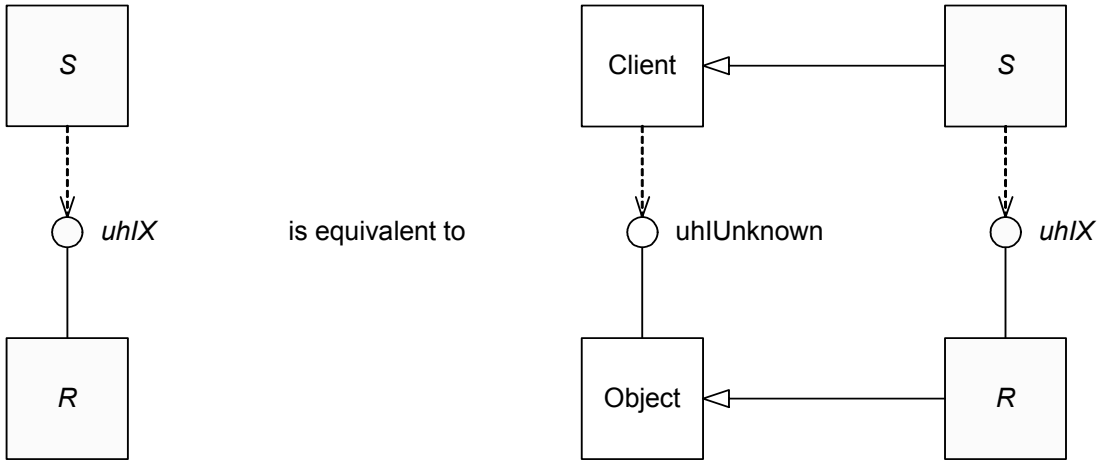


Figure D.1 — Omitting uhIUnknown from Interface-Role Models

— A function of a M3W interface may only be called if the reference count of that interface is greater than zero. In other words, the condition "`refCount` > 0" is part of the precondition of each function, but repeating this condition everywhere is a nuisance. We therefore omit it and assume that the condition "`refCount` > 0" is implicitly contained in the precondition of each M3W function. The only exception is the `uhIUnknown` interface where this condition is made explicit.

— When a M3W function returns a pointer to an interface, it will increment the reference count of the interface before returning; see Annex C. This can be specified explicitly, similar to the way this is done in `QueryInterface`, but repeating this in the specification of each function returning an interface pointer is a nuisance. We therefore omit this and assume that the modification of the reference count is implictly contained in the pre- and post-condition specification of the function.

— In specifying M3W interfaces we omit the `QueryInterface`, `AddRef` and `Release` functions inherited from `uhIUnknown`. They are supposed to be implictly present in each interface and have the properties specified in this document.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— In specifying M3W functions that instantiate objects we should in principle indicate what types of interfaces are provided by the new objects. When the role associated with an object has mandatory interfaces only, this information follows from the interface-role model and is therefore not made explicit. If some interfaces are optional it should be specified which optional interfaces are provided by a specific object and which ones are not. The `iidSet` attribute of Object can be used to specify this.

For example, if a role *R* provides a mandatory interface with IID i_1 and an optional interface with IID i_2, then in the postcondition of a function that creates an instance *x* of *R* we can indicate which interfaces are provided by *x* by specifying

`x.iidSet` **==** { i_1 }

or

`x.iidSet` **==** { i_1, i_2 }.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex E
(informative)

Generation of channel status information

E.1 Non SPDIF Signal Source

In case the input stream of the SPDIF-out component originates from a non SPDIF compliant source, the channel status format is always “consumer-type”. The composition of the channel status information is described in the table below:

Table E.1

Bits	Name	Description
0	pro/con	Defines the Channel status format (consumer or professional). This bit is always set to ‘0’ (only consumer type channel status supported).
1	non-audio	Specifies whether the SPDIF signal contains linear PCM. This property is extracted from the input audio stream.
2	copyright	Depends on `spdifOut.channel[x].copyrightStatus`: `uhSpdifOut_CrUnknown` ‘’ (should never be selected) `uhSpdifOut_CrNone` ‘1’ `uhSpdifOut_CrPreRecorded` ‘0’ `uhSpdifOut_CrHomeCopy` ‘0’
3-5	emphasis	Always set to ‘`000`’.
6-7	channel status mode	Depends on `spdifOut.channel[x].channelStatusMode`.
8-15	category code	Always set to ‘`L0000010`’ (signal processor) where ‘`L`’ depends on `spdifOut.channel[x].copyrightStatus`: `uhSpdifOut_CrUnknown` ‘’ (should never be selected) `uhSpdifOut_CrNone` ‘x’ (value neglected) `uhSpdifOut_CrPreRecorded` ‘1’ `uhSpdifOut_CrHomeCopy` ‘0’
16-19	source number	Depends on `spdifOut.channel[x].sourceNumber`.
20-23	channel number	Depends on `spdifOut.channel[x].channelNumber`.
24-27	sampling frequency	The sampling frequency is extracted from the input audio stream.
28-29	clock accuracy	This property is filled in depending on the capabilities of the underlying hardware.
30-31	reserved	Always set to ‘`00`’.
32	word length (field size)	The word length is extracted from the input audio stream.
33-35	word length	The word length is extracted from the input audio stream.
36-39	reserved	Always set to ‘`0000`’.
40-191	reserved	All bits set to ‘`0`’.

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

E.2 SPDIF Signal Source

In case the source of the SPDIF-out component is a SPDIF signal, the Channel Status data of the input is copied unchanged to the generated output SPDIF signal.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex F
(informative)
Application notes

F.1 Some Commercial Bass Enhancement Algorithms

Barcus Berry Electronics (BBE)

The BBE system addresses the problem of loss of fidelity - or subtle distortions - caused by the inherent characteristics of the loudspeaker by compensating for phase and amplitude distortions (type a)1) subclause 7.2.2.1.1). Varying either the amplitude or the phase of the transients and harmonics within signal causes the desired distortion of the sound's characteristics needed to compensate for the flaws in the speaker characteristics.

(Dynamic) Ultra Bass (DUB, UB)

This feature from Philips gives the impression of deep bass reproduction although the fundamentals are missing, caused by producing harmonics of the low frequency content (type a)2) subclause 7.2.2.1.1). It is used with speaker systems that cannot render sub-woofer frequencies (which is where the fundamental frequencies lie).

Micronas Perfect Bass (MPB, MB)

The Micronas MPB feature is a combination of dynamic bass enhancement and adding harmonics, exploiting the psychoacoustic phenomenon of the ‘missing fundamental’ (type a)2) subclause 7.2.2.1.1). Usage is the same as for Ultra-Bass (Philips).

SRS TruBass (TRUB)

This feature from SRS Labs is based on psychoacoustics and scientific knowledge of the human hearing system. It uses harmonics to deliver deep, rich bass far beyond the physical capabilities of the speaker itself (type a)2) subclause 7.2.2.1.1). When a person listens to audio processed by TruBass, they perceive a heightened sensation of lower bass tones beyond the natural low frequency capabilities of the speaker itself. TruBass technology is customizable to work with any size of speaker.

Dynamic Bass Enhancement (DBE)

This feature from Philips produces a level dependent bass boost. The dynamic behaviour allows a strong bass amplification for small volume signals, but only small bass amplification for high volume signals. This enhancement belongs to the loudness category (type b) in subclause 7.2.2.1.1).

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex G
(informative)

Use of the data Injector in the PVR context

G.1 PVR Application

In the case of a Data Injecting Logical Component instantiated in a PVR-like use-case, one should consider the following clarification on its characteristics and capabilities:

— Output stream

The data that is injected into the platform is streamed out by the Data Injector as soon as the Logical Component further downstream – most probably a demux, or a video/audio decoder - accepts new data. As long as the downstream Logical Component does not ask new data, the data is kept in the buffers.

Thus, a client can effectively control “the use of the injected data” by starting or stopping the downstream component.

The consumption rate (of the injected data) is then determined by the downstream component, provided that enough data is injected by the client.

— I-Frame decoding

In case the Data Injector is used for I-Frame only decoding, the endOfStream parameter in the AddBuffers function must be set to True for each buffer that contains the last part of an I-Frame.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex H
(informative)

Implementation notes

H.1 Instances of the Data Extracting Logical Component

In a system with different use-cases, different instances of the Data Extracting Logical Components with different capabilities may be instantiated for different use-cases. Each instance can have its own specific number, specific alignment, and specific size(s) of buffers, appropriate to the related use-case.

In the same use-case, different instances of the Data Extracting Logical Components may also have different capabilities with respect to the number, the alignment, and the size(s) of the buffers.

H.2 Buffer Management

There are several different situations with respect to buffer management:

— The platform may have specific constraints on the buffers to be used (such as their size(s), their number, their location);

— The platform may require the buffers to be allocated in specific parts of the memory (e.g. because that part is shared memory);

— The platform may also require some specific alignment and/or min/max sizes of the buffers;

— The platform may also have no specific requirements; the client is then free to allocate its own buffers.

To address these situations the uhIDataExtractBuffer interface can be provided to the Data Extracting Client.

However apart from the functionality provided to the run-time client by this interface,

— It is recommended that the platform instance vendor has compile- or boot-time configuration options to allow the customer to set the required number and size of the buffers per Logical Component instance;

— The platform might allocate buffers in client memory and require during integration a “malloc-like” function;

— Static or dynamic allocation of the buffers is up to the platform instance.

H.3 Live viewing/delayed viewing use-case and buffers characteristics

In this use-case, the Middleware either passes content directly from the Data Extractor to the Data Injector (live-viewing move), or it first stores it on HDD and later reads it back and injects it (delayed-viewing mode).

In this single use-case, requirements on the size of the buffers used are rather contradictory:

— in the live-viewing mode, the buffers should better have a relatively small size, so that the performance is acceptable;

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— in the delayed-viewing mode, buffers are better large so that large block(s) of data can be written at once on the Disk.

In this last mode, buffers with a relatively small size are still acceptable when they are contiguous, since writing to a Disk with a large block of data is then possible (considering the data from several buffers in a row) and does not require extra processing/copying; hence the contiguous constraint on the buffers that are managed by the platform.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] ISO/IEC 11172 (all parts), *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s*

[2] ISO/IEC 13818 (all parts), *Information technology – Generic coding of moving pictures and associated audio information*

[3] Digital Video Broadcasting (DVB); Implementation guidelines for the use of MPEG-2 Systems

[4] ISO/IEC 14496-3, *Information technology – Coding of audio-visual objects – Part 3: Audio*

[5] ATSC Standard: Digital Audio Compression Standard (AC-3), Revision A, ATSC Document A/52A

[6] Multichannel digital audio decoding system for consumer products - Licensee information manual

[7] DTS Coherent Acoustics Decoder, version 1.1, September 1998.

[8] Television Systems; NICAM 728: transmission of two-channel digital sound with terrestrial television

[9] Multichannel TV Sound System BTSC System Recommended Practices, CEA TVSB5.

[10] Transmission of multisound in terrestrial television systems PAL B, D1, G, H and I, and SEC

[11] Specifications of the Macrovision Copy Protection Process for Authorised Component Supplier

[12] Multichannel digital audio decoding system for consumer products – Licensee information manual

[13] ATSC Standard: Digital Television Standard, Revision B, ATSC Document A/53B, August 2001.

[14] Recommended Practice for Line 21 Data Service, ANSI/EIA 608-94.

[15] Digital Television (DTV) Closed Captioning, EIA-708-B.

[16] Home Networked Device interoperability Guidelines version 1.0, Digital Living Network Alliance

[17] High-Definition Multimedia Interface Specification 1.0

[18] EIA, EIA/CEA-861B, A DTV Profile for Uncompressed High Speed Digital Interfaces

[19] IEC 61937 (all parts): interface for non-PCM encoded Audio bitstreams

[20] Licensee Information Manual: Dolby Digital Consumer Decoder. Issue 5.

[21] IEC 60958 (all parts), *Digital audio interface*

[22] Dolby licensee information manual: Dolby digital consumer decoder, issue 5, Dolby Laboratories

[23] Digital Video Broadcasting (DVB); Support for use of scrambling and Conditional Access

[24] Data Encription Standard (DES), U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology

[25] DES Modes of Operation, National Institute of Standards and Technology

[26] Detailed Functional Specification of the MPEG and Data Processing Unit for the 2nd Generation d-box

© ISO/IEC 2007 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[27] The Component Object Model Specification, Version 0.9, Microsoft Corporation (1995).

[28] Inside COM, D. Rogerson, Microsoft Press (1997).

[29] URL specification, http://www.w3.org

[30] Television systems; Specification of the domestic video Programme Delivery Control system

[31] Television systems; 625-line television Wide Screen Signalling (WSS), EN 300 294 V1.3.2.

[32] Television systems; Data transmission within Teletext, ETS 300 708.

[33] Video Demystified, Keith Jack.

[34] "System API" Gemstar US.

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ICS 35.040

Price based on 383 pages

© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale