
**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)

AVA Documents
Document Sharing Hub
© ISO/IEC 2007

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

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

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*

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.

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*

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

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

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.

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

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

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

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

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**LTl**

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

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

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

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 derivatives 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

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

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 uhlUnknown 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 uhlUnknown 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

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**VS**

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

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
uhlConnMgr	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.
uhlFatErr	Fatal Error	This API specification defines the interfaces associated with the fatal and non-fatal error handling.
uhllamAlive	I am alive	This API specification defines the I am Alive interface suite which consists of the interfaces uhllamAlive and uhllamAliveNtf, that support “watchdog” functionality.
uhlOutConf	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.

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.
rclUnknown / uhIUnknown	IUnknown	This API specification defines the rclUnknown interface which is the M3W equivalent of the IUnknown interface of Microsoft COM. The rclUnknown 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
uhlAnaAdec	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.
uhlAtscDec	ATSC Decoder	This component is responsible for demultiplexing and decoding an ATSC compliant Transport Stream (TS) into its constituent components.
uhlSpdifDec	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.
uhlSpdifIn	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.
uhlSpdifOut	SPDIF-out	Transforms various formats of a digital audio stream to a SPDIF IEC compliant stream.

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 output 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.

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.
UhlHdmIn	HdmIn	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.

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.

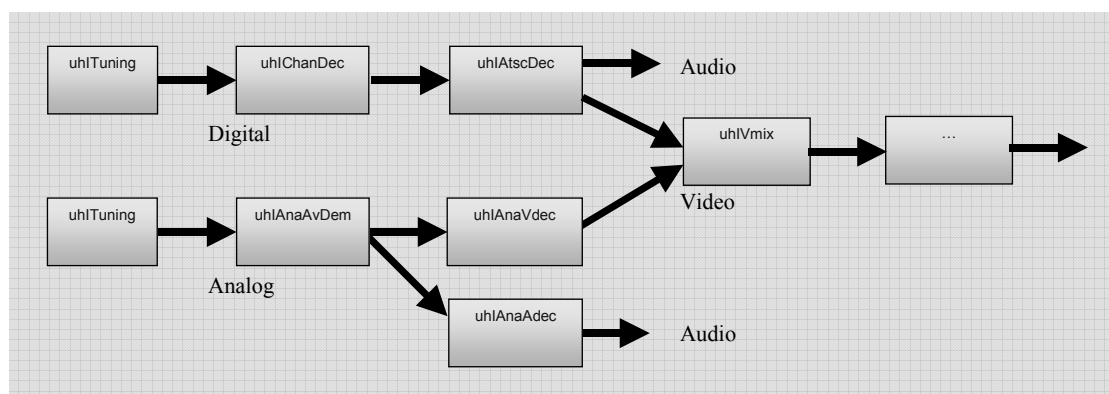


Figure 1 — Simplified example graph.

Table 5 — Description of the video Processing & Rendering.

Specification	Name	Description
uhlAmbientLevel	Ambient Level	Set the measured ambient light. This can then be used by the platform to optimise the video processing algorithms based on this.
UhlAnaVenc	Analog Video Encoder	The Analog Video Encoder gives control over the encoding of a digital video stream into an analog video signal.
uhlAnaVencrypt	Analog Video Encryption	The Analog Video Encryptor gives control over an anti-taping feature.
uhlAntiAging	Anti Aging	Control the algorithm used to prevent aging in case of static images being displayed (e.g. logo's).
UhlBbarDet	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.
UhlColEnh	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.
UhlCti	Color Transient Improvement	Allows control over the color transient improvement functionality (to increase the perceived color sharpness).
UhlDnr	Dynamic Noise Reduction	To control the reduction of the noise in a video stream.
uhlHistoMod	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.
uhlNoiseMeas	Noise Measurement	The Noise Measurement offers an abstract view on the amount of noise in a video signal.
uhlScanRateConv2	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).
uhlSharpEnh	Sharpness Enhancement	Sharpness Enhancement deals with enhancing the sharpness of the luminance (Y) component of a YUV signal.
uhlSharpMeas	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).
UhlVfeat	Basic Video Featuring	Basic Video Featuring represents functionality to control the brightness, contrast, saturation, hue, and white point of a video signal.
UhlVmix	Video Mixing	Video Mixer represents functionality to compose a single video stream from multiple inputs, each of which can either be video or graphics.

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

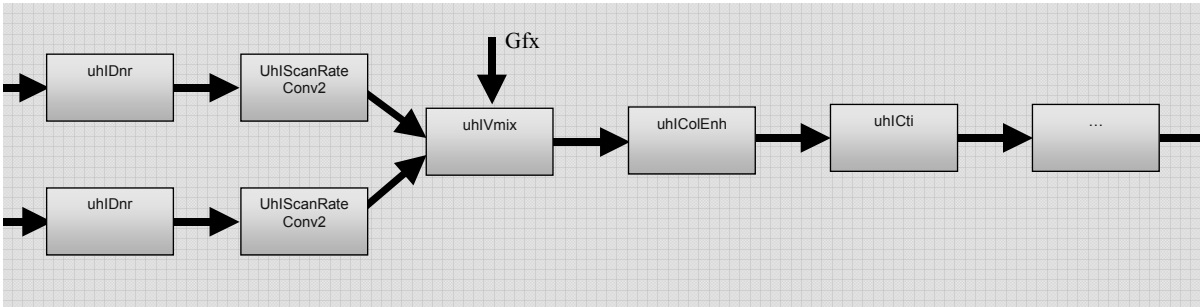


Figure 2 — Possible Video processing & Rendering graph.

Table 6 — Description of the personal Video Recording related.

Specification	Name	Description
uHlDataInject	Data Injecting	The Data Injecting component allows the injection of streaming data into the platform. This can be used for a PVR application.
uHlDataExtract	Data Extracting	The Data Extracting component allows the extraction of streaming data from the platform. This can be used for a PVR application.
uHlSptsTrMux	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.

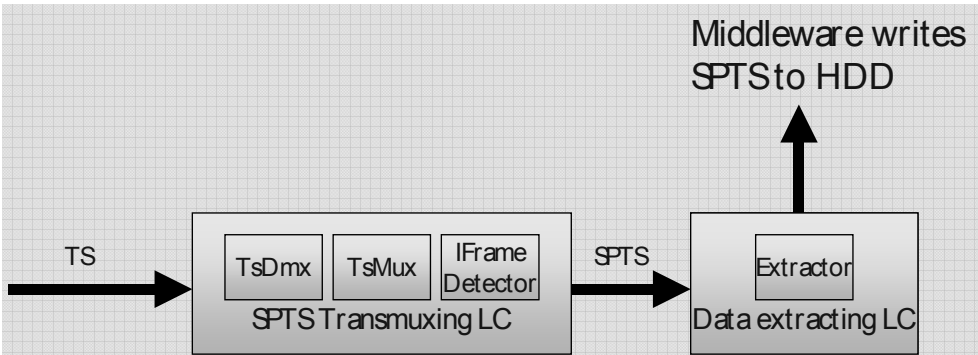


Figure 3 — Extracting a Single Program Transport Stream for recording.

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.

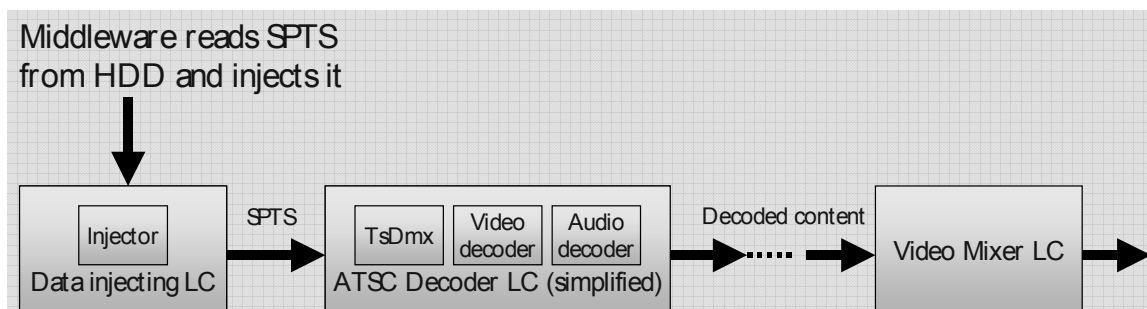


Figure 4 — Injecting a Single Program Transport Stream for playback.

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).

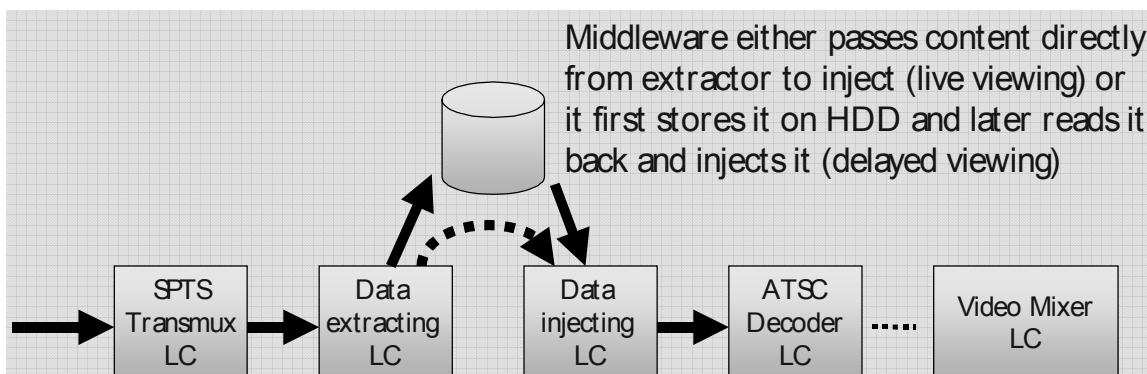


Figure 5 — Combining Data Extraction and Injection for delayed viewing.

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.

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 uhlConnMgr

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).

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.

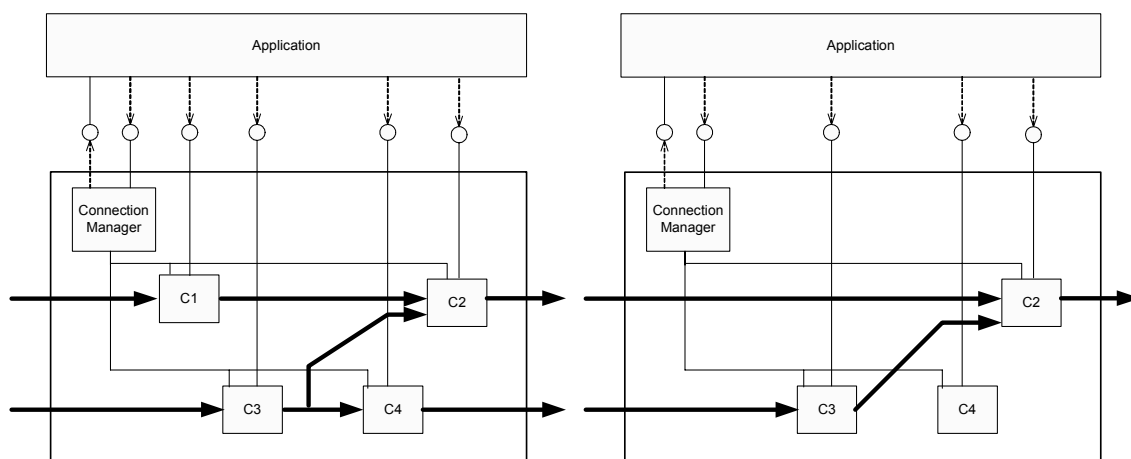


Figure 6 — Example: Switch between two use cases

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.)

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 <code>GetInstance</code> function.

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 `uhConnMgrPlf_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.

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

Description

Values of this type represent sets of notification functions which are typically constructed by logical OR-ing of values of type `uhConnMgrPlf_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 `uhConnMgrPlf_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
<code>UseCaseSelectionInProgress</code>	Selection of a use case is in progress.
<code>UseCaseSelectionCompleted</code>	Selection of a use case is completed.

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 <code>GetInstance</code> 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.

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
<code>from</code>	Use case from which the transition should be made (i.e. the use case that is current when the transition is started).
<code>to</code>	Use case to which the transition should be made (i.e. the selected use case).

Constraints

None.

6.1.1.3 Logical Component

6.1.1.3.1 Interface-Role Model

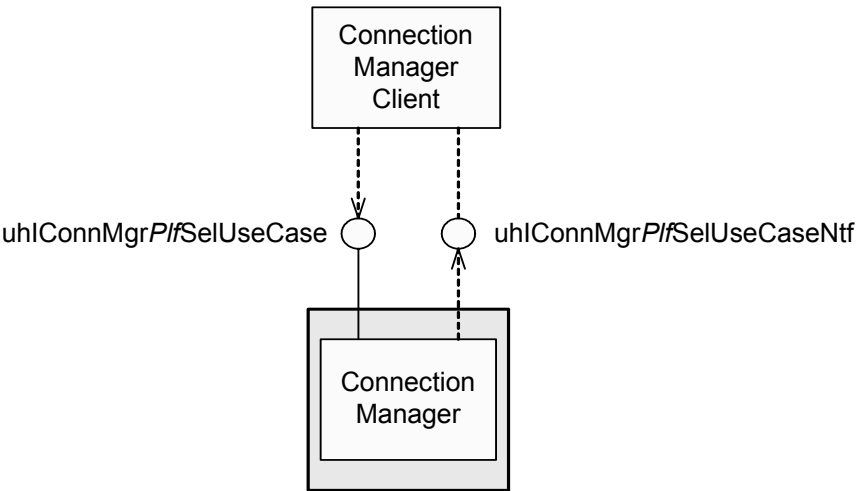


Figure 7 — Interface Role Model

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
<code>ConnectionManager</code>	<code>uhIConnMgrPlfSelUseCase</code>	mandatory
<code>ConnectionManagerClient</code>	<code>uhIConnMgrPlfSelUseCaseNtf</code>	optional ¹

¹The `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.

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".

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
<code>useCases</code>	The set of use cases supported by the platform instance. The value of <code>useCases</code> is constant because it is set at instantiation time and does not change dynamically. For every platform instance this set contains the <code>Null use case</code> .
<code>allowedTransitions</code>	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 <code>allowedTransitions</code> is constant because it is set at instantiation time and does not change dynamically.
<code>currentUseCase</code>	The use case that is currently selected. This attribute is only valid when the state is equal to <code>UseCaseSelectionCompleted</code> .
<code>state</code>	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.

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.

6.1.1.5.1.1 Subscribe

Signature

```
uhErrorCode_t Subscribe (  
    [in] uhTConnMgrPlfSelUseCaseNtf    *pINotify,  
    [in] UInt32                        cookie,  
    [in] uhConnMgrPlf_Notifications_t  notif);
```

Qualifiers

— subscribe-function

6.1.1.5.1.2 Unsubscribe

Signature

```
uhErrorCode_t Unsubscribe (  
    [in] uhTConnMgrPlfSelUseCaseNtf    *pINotify,  
    [in] UInt32                        cookie,  
    [in] uhConnMgrPlf_Notifications_t  notif);
```

Qualifiers

— unsubscribe-function

6.1.1.5.1.3 SelectUseCaseX

Signature

```
uhErrorCode_t SelectUseCaseX (...);
```

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.

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

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

Return Values

Table 23

Name	Description
<code>uh_OK</code>	Indicates that the component instance provides the requested interface.
<code>uh_ERR_PLF_NO_INTERFACE</code>	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)`

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
<code>useCaseId</code>	Identifies the use case
<code>pComps0to31</code>	Pointer to a bit array in which bit <i>i</i> indicates whether the component with <code>uhConnMgr_CompId_t</code> equal to <i>i</i> is contained in the specified use case.
<code>pComps32to63</code>	Pointer to a bit array in which bit <i>i</i> indicates whether the component with <code>uhConnMgr_CompId_t</code> equal to (<i>i</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`

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

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
<code>cookie</code>	The cookie identifying the subscription to which this notification applies.
<code>useCaseId</code>	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

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.

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.

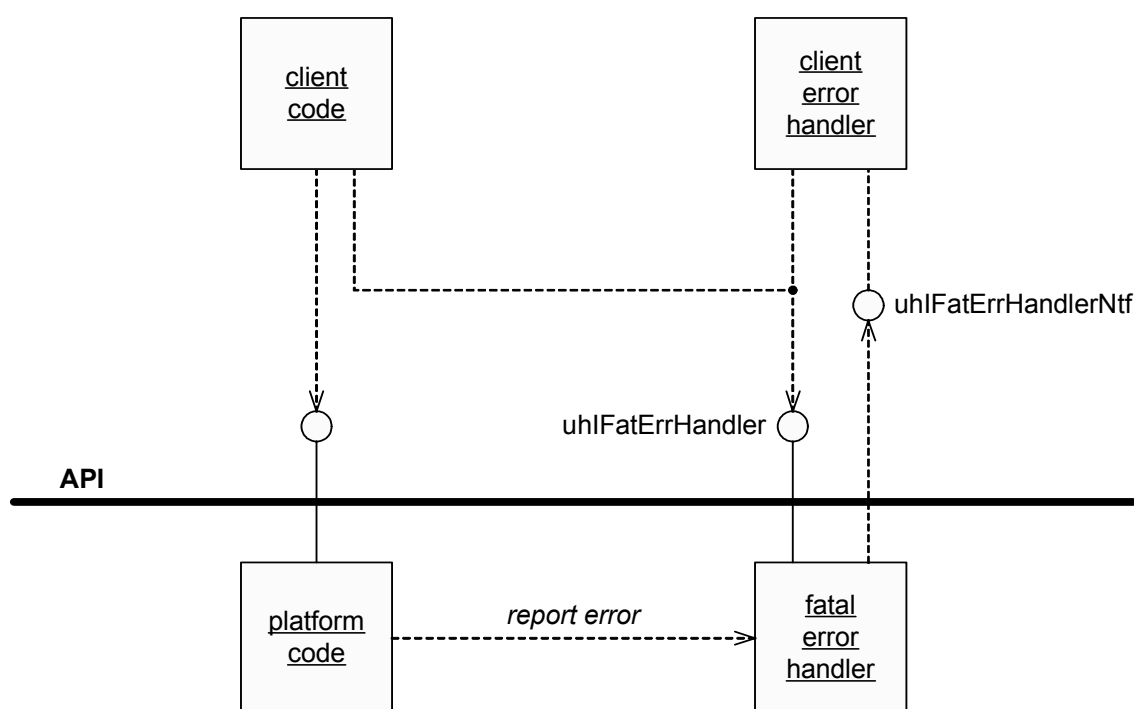


Figure 8 — Typical Fatal Error Handler Configuration

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.

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.

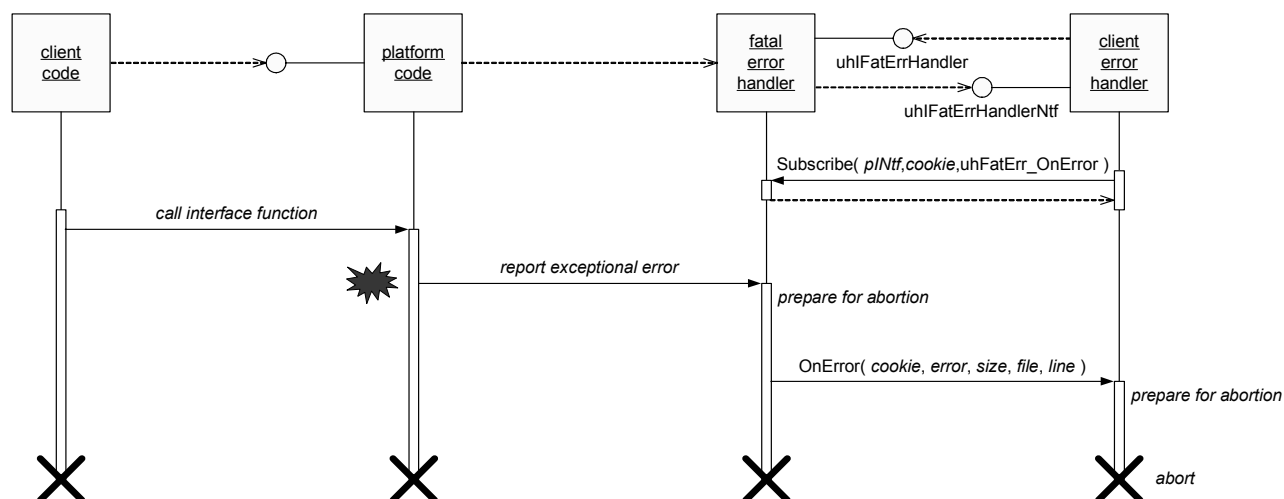


Figure 9 — Fatal Error Handling Protocol with installed Client Error Handler