OpenMAX™ Application Layer Application Programming Interface Specification

Version 1.1 Specification

January 18, 2011

Table of Contents

PART 1: USER MANUAL 1
1 OVERVIEW 2
1.1 PURPOSE OF THIS DOCUMENT 2
1.1.1 About the Khronos Group 2
1.2 SCOPE 2
1.3 INTENDED AUDIENCE 2
1.4 A BRIEF HISTORY OF OPENMAX 3
1.4.1 The OpenMAX Application Layer 3
1.4.2 Relationship to OpenMAX IL 4
1.5 RELATIONSHIP TO OPENSL ES 4
1.6 CONVENTIONS USED 6
1.6.1 Parameter Range Notation 6
1.6.2 Format and Typographic Conventions 7
1.7 ACKNOWLEDGEMENTS 8
2 OPENMAX AL FEATURES AND PROFILES 9
2.1 MOTIVATION 9
2.2 OPENMAX AL PROFILE DEFINITION 9
2.3 PROFILES 9
2.4 OPTIONALITY RULES OF FEATURES AND PROFILES 12
2.5 MIDI IN OPENMAX AL 13
2.6 PROFILE NOTES 13
2.7 BEHAVIOR FOR UNSUPPORTED FEATURES 13
3 DESIGN OVERVIEW 14
3.1 OBJECT MODEL 14
3.1.1 Objects and Interfaces 14
3.1.2 Getters and Setters 15
3.1.3 Representation in Code 16

3.1.4 The XAObjectItf Interface *16*
3.1.5 The Engine Object and XAEngineItf Interface *17*
3.1.6 The Relationship Between Objects and Interfaces *17*
3.1.7 The XADynamicInterfaceManagementItf Interface *19*
3.1.8 Resource Allocation *19*
3.2 THREADING MODEL 20
3.2.1 Mode of Operation *20*
3.2.2 Thread Safety *21*
3.3 NOTIFICATIONS 21
3.4 ERROR REPORTING 22
3.5 MEMORY MANAGEMENT 22
3.5.1 General *22*
3.5.1.1 Parameters 22
3.5.2 Callbacks *23*
3.5.3 Exceptions *23*
3.6 EXTENSIBILITY 23
3.6.1 Principles *23*
3.6.2 Permitted Modifications to Physical Code *23*
3.6.3 Extending Supported Interface Types *23*
3.6.4 Extending Supported Object Types *24*
3.6.5 Extending Method Parameter Ranges *24*
3.6.6 Result Codes *24*
3.6.7 Data Locators *24*
3.6.8 Interface ID Allocation Scheme *25*
3.6.9 Avoiding Naming Collisions *25*
4 FUNCTIONAL OVERVIEW 27
4.1 OBJECT OVERVIEW 27

4.1.1 Engine Object *27*
4.1.2 Media Objects *28*
4.1.2.1 Data Source and Sink Structures 28
4.1.3 Metadata Extractor Object *29*
4.1.4 Audio Output Mix Object *29*
4.1.5 Camera Object *29*
4.1.6 LED Array Control Object *29*
4.1.7 Radio Object *29*
4.1.8 Vibration Control Object *30*
4.2 AUDIO PLAYBACK AND RECORDING 30
4.3 VIDEO PLAYBACK AND RECORDING 30
4.4 IMAGE RENDERERING AND CAPTURE 30
4.5 PLAYBACK OF MIDI 30
4.5.1 Support for Mobile DLS *31*
4.6 DISPLAY REGIONS 31
4.7 OPENMAX AL USE CASES 33
4.7.1 Audio and Video Playback *33*
4.7.2 Audio Playback *34*
4.7.3 Recording Audio *35*
4.7.4 Image Player *36*
4.7.5 Video Camera *37*
4.7.6 Still Camera *38*
4.7.7 Radio Playback *39*
4.7.8 Reading Metadata *40*
PART 2: API REFERENCE 41
5 BASE TYPES AND UNITS 42
5.1 STANDARD UNITS 42
5.2 BASE TYPES 42

6 FUNCTIONS 44

6.1 XACREATEENGINE 44

6.2 XAQUERYNUMSUPPORTEDENGINEINTERFACES 46

6.3 XAQUERYSUPPORTEDENGINEINTERFACES 46

7 OBJECT DEFINITIONS 47

7.1 CAMERA I/O DEVICE 48

7.2 ENGINE OBJECT 49

7.3 LED ARRAY I/O DEVICE 51

7.4 MEDIA PLAYER OBJECT 52

7.5 MEDIA RECORDER OBJECT 56

7.6 METADATA EXTRACTOR OBJECT 59

7.7 OUTPUT MIX OBJECT 61

7.8 RADIO I/O DEVICE 63

7.9 VIBRA I/O DEVICE 64

8 INTERFACE DEFINITIONS 65

8.1 XA_IID_NULL 66

8.2 XAAUDIODECODERCAPABILITIESITF 67

8.3 XAAUDIOENCODERITF 70

8.4 XAAUDIOENCODERCAPABILITIESITF 72

8.5 XAAUDIOIODEVICECAPABILITIESITF 75

8.6 XACAMERAITF 89

8.7 XACAMERACAPABILITIESITF 113

8.8 XACONFIGEXTENSIONSITF 126

8.9 XADEVICEVOLUMEITF 129

8.10 XADYNAMICINTERFACEMANAGEMENTITF 132

8.11 XADYNAMICSOURCEITF 137

8.12 XADYNAMICSOURCESINKCHANGEITF 139

8.13 XAENGINEITF 146

8.14 XAEqualizerItf 168
8.15 XAImageControlsItf 178
8.16 XAImageDecoderCapabilitiesItf 184
8.17 XAImageEffectsItf 187
8.18 XAImageEncoderItf 191
8.19 XAImageEncoderCapabilitiesItf 194
8.20 XALEDArrayItf 197
8.21 XAMetadataExtractionItf 201
8.22 XAMetadataInsertionItf 213
8.23 XAMetadataMessageItf 224
8.24 XAMetadataTraversalItf 226
8.25 XAObjectItf 231
8.26 XAOutputMixItf 241
8.27 XAPlayItf 245
8.28 XAPlaybackRateItf 255
8.29 XAPrefetchStatusItf 262
8.30 XARadioItf 268
8.31 XARDSItf 283
8.32 XARecordItf 301
8.33 XASeekItf 310
8.34 XASnapshotItf 314
8.35 XAStreamInformationItf 324
8.36 XAThreadSyncItf 333
8.37 XAVibraItf 335
8.38 XAVideoDecoderCapabilitiesItf 340
8.39 XAVideoEncoderItf 343
8.40 XAVideoEncoderCapabilitiesItf 345

8.41 XAVideoPostProcessingItf 348
8.42 XAVolumeItf 356
9 MACROS AND TYPEDEFS 363
9.1 Structures 364
9.1.1 XAAudioCodecDescriptor 364
9.1.2 XAAudioEncoderSettings 366
9.1.3 XAAudioInputDescriptor 367
9.1.4 XAAudioOutputDescriptor 369
9.1.5 XAAudioStreamInformation 370
9.1.6 XACameraDescriptor 371
9.1.7 XADataFormat_MIME 372
9.1.8 XADataFormat_PCM 372
9.1.9 XADataFormat_PCM_EX 374
9.1.10 XADataFormat_RawImage 375
9.1.11 XADataLocator_Address 375
9.1.12 XADataLocator_ContentPipe 375
9.1.13 XADataLocator_IODevice 376
9.1.14 XADataLocator_MediaObject 377
9.1.15 XADataLocator_NativeDisplay 377
9.1.16 XADataLocator_Null 377
9.1.17 XADataLocator_OutputMix 377
9.1.18 XADataLocator_URI 378
9.1.19 XADataSink 379
9.1.20 XADataSource 380
9.1.21 XAEngineOption 381
9.1.22 XAFocusPointPosition 381
9.1.23 XAHSL 381

9.1.24 XAImageCodecDescriptor *382*
9.1.25 XAImageSettings *382*
9.1.26 XAImageStreamInformation *382*
9.1.27 XAInterfaceID *384*
9.1.28 XALEDDescriptor *384*
9.1.29 XAMediaContainerInformation *384*
9.1.30 XAMetadataInfo *385*
9.1.31 XAMIDIStreamInformation *385*
9.1.32 XANativeHandle *386*
9.1.33 XARectangle *386*
9.1.34 XATimedTextStreamInformation *387*
9.1.35 XAVendorStreamInformation *387*
9.1.36 XAVibraDescriptor *388*
9.1.37 XAVideoCodecDescriptor *388*
9.1.38 XAVideoSettings *389*
9.1.39 XAVideoStreamInformation *390*
9.2 MACROS 391
9.2.1 XA_API *391*
9.2.2 XAAPIENTRY *391*
9.2.3 XA_AUDIOCODEC *391*
9.2.4 XA_AUDIOPROFILE and XA_AUDIOMODE *391*
9.2.5 XA_BOOLEAN *397*
9.2.6 XA_BYTEORDER *398*
9.2.7 XA_CAMERA_APERTUREMODE *398*
9.2.8 XA_CAMERA_AUTOEXPOSURESTATUS *398*
9.2.9 XA_CAMERACBEVENT *399*
9.2.10 XA_CAMERACAP *400*

9.2.11 XA_CAMERA_EXPOSUREMODE 401
9.2.12 XA_CAMERA_FLASHMODE 402
9.2.13 XA_CAMERA_FOCUSMODE 403
9.2.14 XA_CAMERA_FOCUSMODESTATUS 404
9.2.15 XA_CAMERA_ISOSENSITIVITYMODE 404
9.2.16 XA_CAMERA_LOCK 404
9.2.17 XA_CAMERA_METERINGMODE 405
9.2.18 XA_CAMERA_SHUTTERSPEEDMODE 405
9.2.19 XA_CAMERA_WHITEBALANCEMODE 406
9.2.20 XA_CAMERA_ZOOM 407
9.2.21 XA_CHARACTERENCODING 407
9.2.22 XA_COLORFORMAT 409
9.2.23 XA_CONTAINERTYPE 412
9.2.24 XA_DATAFORMAT 413
9.2.25 XA_DATALOCATOR 414
9.2.26 XA_DEFAULTDEVICEID 414
9.2.27 XA_DEVICECONNECTION 415
9.2.28 XA_DEVICELOCATION 416
9.2.29 XA_DEVICESCOPE 416
9.2.30 XA_DOMAINTYPE 417
9.2.31 XA_DYNAMIC_ITF_EVENT 417
9.2.32 XA_ENGINEOPTION 418
9.2.33 XA_EQUALIZER 419
9.2.34 XA_FOCUSPOINTS 420
9.2.35 XA_FREQRANGE 423
9.2.36 XA_IMAGECODEC 423
9.2.37 XA_IMAGEEFFECT 424

9.2.38 XA_IODEVICE 424
9.2.39 XA_METADATA_FILTER 425
9.2.40 XA_METADATATRAVERSALMODE 425
9.2.41 XA_MIDIBANK 426
9.2.42 XA_MIDI_UNKNOWN 426
9.2.43 XA_MILLIBEL 426
9.2.44 XA_MILLIHERTZ_MAX 426
9.2.45 XA_MILLIMETER_MAX 426
9.2.46 XA_NODE_PARENT 427
9.2.47 XA_NODETYPE 427
9.2.48 XA_OBJECT_EVENT 428
9.2.49 XA_OBJECT_STATE 428
9.2.50 XA_OBJECTID 429
9.2.51 XA_ORIENTATION 429
9.2.52 XA_PCM_REPRESENTATION 430
9.2.53 XA_PCMSAMPLEFORMAT 430
9.2.54 XA_PLAYEVENT 430
9.2.55 XA_PLAYSTATE 431
9.2.56 XA_PREFETCHEVENT 431
9.2.57 XA_PREFETCHSTATUS 432
9.2.58 XA_PRIORITY 432
9.2.59 XA_PROFILE 432
9.2.60 XA_RADIO_EVENT 434
9.2.61 XA_RATECONTROLMODE 436
9.2.62 XA_RATEPROP 436
9.2.63 XA_RDS_EVENT 437
9.2.64 XA_RDSPROGRAMMETYPE 438

9.2.65 XA_RDSRTPLUS 443
9.2.66 XA_RECORDEVENT 446
9.2.67 XA_RECORDSTATE 446
9.2.68 XA_RENDERINGHINT 447
9.2.69 XA_RESULT 447
9.2.70 XA_ROOT_NODE_ID 448
9.2.71 XA_SAMPLINGRATE 449
9.2.72 XA_SEEKMODE 449
9.2.73 XA_STEREO_MODE 450
9.2.74 XA_SPEAKER 450
9.2.75 XA_STREAMCBEVENT 451
9.2.76 XA_TIME 451
9.2.77 XA_VIDEOCODEC 452
9.2.78 XA_VIDEOMIRROR 452
9.2.79 XA_VIDEOPROFILE and XA_VIDEOLEVEL 453
9.2.80 XA_VIDEOSCALE 459
PART 3: APPENDICES 460
APPENDIX A: REFERENCES 461
APPENDIX B: GLOSSARY OF RDS TERMS 462
APPENDIX C: OBJECT-INTERFACE MAPPING 463
APPENDIX D: SAMPLE CODE 466
D.1 AUDIO PLAYBACK WITH EQUALIZER 466
D.2 AUDIO/VIDEO PLAYBACK 470
D.3 RADIO WITH RDS SUPPORT 476
D.4 AUDIO RECORDING THROUGH MICROPHONE 481
D.5 SNAPSHOT WITH PREVIEW 485
D.6 METADATA EXTRACTION 492

PART 1: USER MANUAL

1 Overview

1.1 Purpose of this Document

This document details the API for OpenMAX Application Layer (AL) 1.1. Developed as an open standard by the Khronos Group, OpenMAX AL™ is an application-level multimedia playback and recording API for mobile embedded devices. It provides a device-independent, cross-platform interface for applications to access a device's audio, video and imaging capabilities.

1.1.1 About the Khronos Group

The Khronos Group is a member-funded industry consortium focused on the creation of open standard, royalty-free APIs to enable the authoring and accelerated playback of dynamic media on a wide variety of platforms and devices. All Khronos members can contribute to the development of Khronos API specifications, are empowered to vote at various stages before public deployment, and may accelerate the delivery of their multimedia platforms and applications through early access to specification drafts and conformance tests. The Khronos Group is responsible for open APIs such as OpenGL® ES, OpenKODE™, OpenSL ES™ and OpenVG™.

1.2 Scope

OpenMAX AL accommodates common multimedia application use cases by standardizing a set of representative objects, as well as interfaces on those objects, to control and configure them.

It is an application-level, C-language, multimedia API designed for resource-constrained devices. The OpenMAX AL API design puts particular emphasis on ensuring the API is suitable for mobile embedded devices - including basic mobile phones, smart "feature" phones, PDAs and mobile digital music players. Nevertheless, this does not preclude its applicability to other sophisticated media playback and recording devices.

The OpenMAX AL API design devotes particular attention to application-developer friendliness. Its status as an open cross-platform API enables developers to port the same source across multiple devices with minimal effort. Thus OpenMAX AL provides a stable base for application development.

This document specifies the OpenMAX AL API. It does not define how to implement the API.

1.3 Intended Audience

This specification is meant for application-developers and implementers. The document is split into a user manual section and an API reference section. Application-developers can use this document as a user guide to learn about how to use OpenMAX AL and they can refer to the API reference when developing their applications. Implementers of the API can use this specification to determine what constitutes an implementation conforming to the OpenMAX AL standard.

1.4 A Brief History of OpenMAX

The OpenMAX set of APIs was originally conceived as a method of enabling portability of components and media applications throughout the mobile device landscape. Brought into the Khronos Group in mid-2004 by a handful of key mobile hardware companies, OpenMAX has gained the contributions of companies and institutions spanning the spectrum of the multimedia field. As such, OpenMAX stands to unify the industry in taking steps toward media component portability. Stepping beyond mobile platforms, the general nature of the OpenMAX AL API makes it applicable to all media platforms.

1.4.1 The OpenMAX Application Layer

The OpenMAX AL API provides application-level, multimedia solutions with portability across an array of platforms - by providing a common abstraction for a system's media playback and recording functionality. The API organizes this abstraction around a set of high level objects. An application acquires all objects from one "engine" object which encapsulates an OpenMax AL session and serves as an umbrella for all other objects.

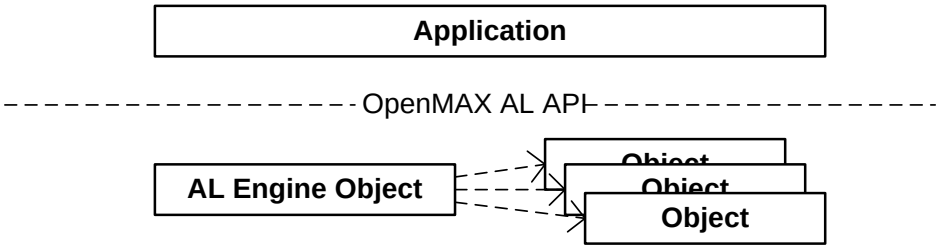


Figure 1: OpenMAX AL application, engine and object abstraction

Principle among these objects is the "media object". Each media object represents either a playback or recording task, denoted media player or media recorder respectively, which takes data from a designated source and sends it to a designated sink.

In some cases, the sources and sinks are themselves objects. For instance a camera device might act as a source or an audio output mix might act as a sink. In other cases a source or sink is simply a location, such as a memory buffer containing a sound or a file to which one writes recorded data.

All objects expose interfaces which serve as controls relevant to their operation. Interfaces constitute structures of methods grouped by functional affinity. Thus a player exposes interfaces for playback, rate control, seeking and metadata extraction.

An application constructs a use case by instantiating the requisite set of objects and then creating the correct associations between them. For instance, an application achieves playback of a 3gp file by creating an audio output mix and then creating a media player with that output mix as an audio sink, a video window as the video sink, and a file as the source.

The application controls a use case by retrieving interfaces from the objects that implement it and calling methods on those interfaces. Given the playback example above, the application may retrieve a playback interface from the player and then control the player's operation by using methods on the playback interface to change the state between stopped, paused and playing.

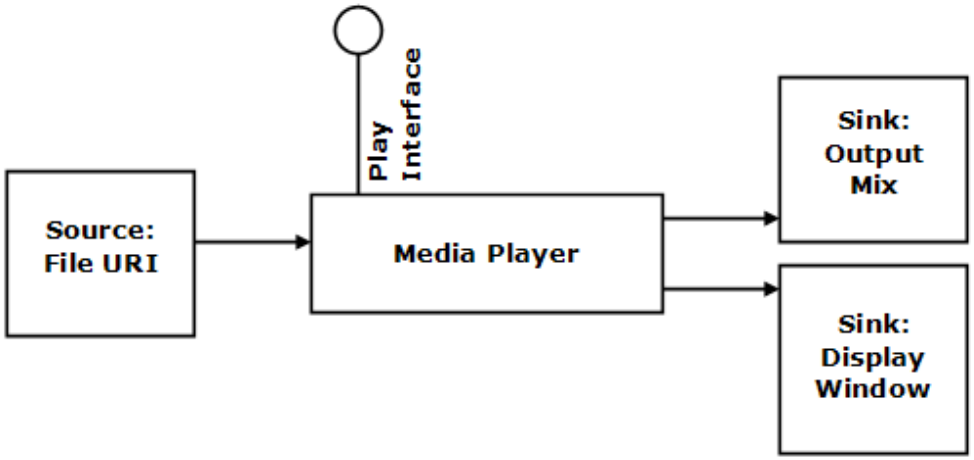


Figure 2: Local file playback use case

1.4.2 Relationship to OpenMAX IL

The OpenMAX AL represents the highest layer of the OpenMAX family of APIs. As such, it serves the multimedia needs of application developers.

The Khronos group also provides an API for system integrators, denoted as the OpenMAX Integration Layer (IL). OpenMAX IL defines an integration framework for the internals of a multimedia architecture which abstracts the codecs, file manipulations, transformations, and peripheral components installed on a system. The IL also provides a means for these components to interoperate with each other - even if they are delivered from multiple sources.

The working groups for each API have co-operated to deliberately design AL to be amenable to an IL-based implementation. For example, IL defines the set of low-level components to satisfy the constituent functionality of high-level AL use cases. Thus, an AL implementor may construct a media object as a chain of IL components.

OpenMAX AL does not mandate an AL solution be based on IL (nor does it mandate any particular implementation detail). Nevertheless, the relationship between APIs enables a rich and efficient software ecosystem for multimedia.

1.5 Relationship to OpenSL ES

OpenMAX AL is an application-level multimedia playback and recording API for mobile embedded devices. OpenSL ES [OSLES] is an application-level enhanced audio API, also designed for mobile embedded devices. As such, both APIs do overlap in certain basic audio functionality (such as audio playback, audio recording and basic MIDI). The Venn diagram in Figure 3 illustrates the functional overlap in the two APIs.

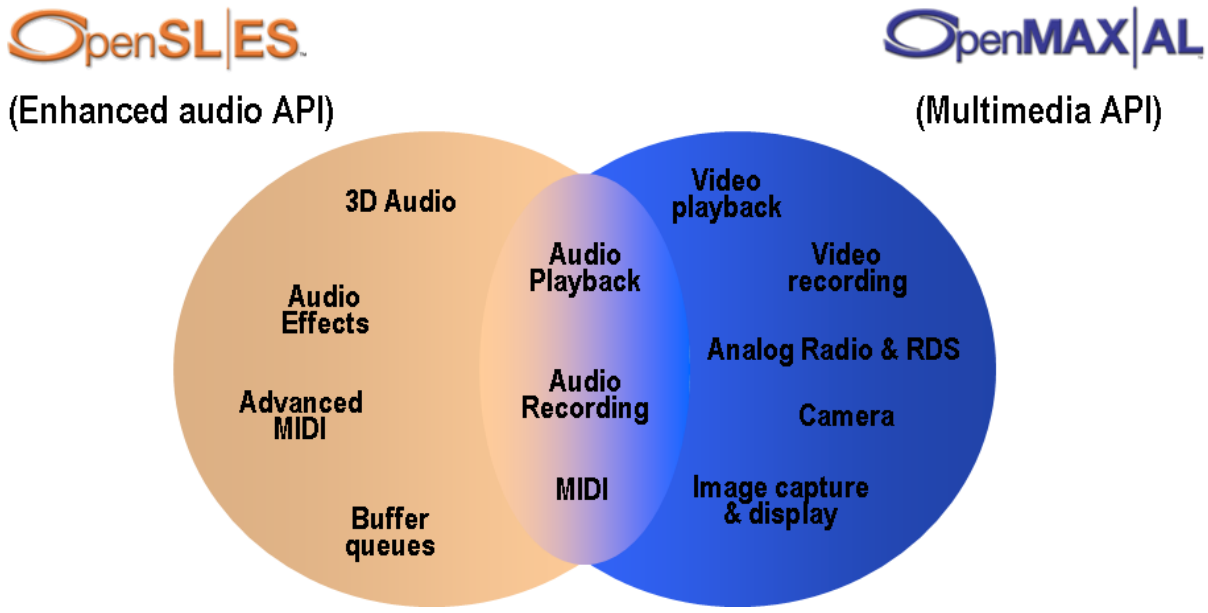


Figure 3: OpenSL ES versus OpenMAX AL

As the Venn diagram shows, OpenMAX AL has audio features like analog radio tuner and RDS that are not part of OpenSL ES. Similarly, OpenSL ES has advanced audio features like effects (reverberation, stereo widening, bass boost, etc.) and positional 3D audio that are not part of OpenMAX AL.

The primary focus of OpenMAX AL is media (audio, video, and image) capture and rendering. The primary focus of OpenSL ES is advanced audio and MIDI functionality for mobile devices. Further, both OpenMAX AL and OpenSL ES are partitioned into profiles based on market segments:

- OpenSL ES has three overlapping profiles: Phone, Music and Game.
- OpenMAX AL has two overlapping profiles: Media Player and Media Player/Recorder.

Each of these profiles has well-defined feature sets and conformance requirements. For example, to be compliant with the OpenMAX AL Media Player profile, an OpenMAX AL implementation must provide audio, image and video playback functionality. An audio-only OpenMAX AL implementation would not be compliant with either profile of the OpenMAX AL specification.

This segmentation into profiles ensures that there will be no confusion whatsoever regarding which API is suitable for a particular set of use cases.

- Example 1: an audio-only application will have no need for video and image functionality and therefore would likely pick one of the OpenSL ES profiles, depending on the use cases of interest.
- Example 2: a media playback and recording application would use the OpenMAX AL Media Player/Recorder profile.
- Example 3: An advanced multimedia/game application that needs audio/video/image playback and recording as well as advanced audio features like 3D audio and effects would use both the Media Player/Recorder profile of OpenMAX AL and the Game profile of OpenSL ES.

The two APIs have been designed such that their architecture is identical. Further, each API has identical API methods for the same functionality. At the same time, the APIs are also independent – each can be used as a

standalone API by itself (as in Examples 1 and 2) or can co-exist with the other on the same device (as in Example 3).

1.6 Conventions Used

When this specification discusses requirements and features of the OpenMAX AL API, specific words are used to indicate the requirement of a feature in an implementation. The table below shows a list of these words.

Table 1: Requirement Terminology

Word	Definition
May	The stated functionality is an optional requirement for an implementation of the OpenMAX AL API. Optional features are not required by the specification but may have conformance requirements if they are implemented. This is an optional feature as in "The implementation *may* have vendor-specific extensions."
Shall	The stated functionality is a requirement for an implementation of the OpenMAX AL API. If an implementation fails to meet a *shall* statement, it is not considered as conforming to this specification. *Shall* is always used as a requirement, as in "The implementation *shall* support the play interface".
Should	The stated functionality is not a requirement for an implementation of the OpenMAX AL API but is recommended or is a good practice. *Should* is usually used as follows: "An OpenMAX AL implementation of the game profile *should* be capable of playing content encoded with an MP3 codec". While this is a recommended practice, an implementation could still be considered as conforming to the OpenMAX AL API without implementing this functionality.
Will	The stated functionality is not a requirement for an implementation of the OpenMAX AL API. *Will* is typically used when referring to a third party, as in "the application framework *will* correctly handle errors".
Deprecated	Some APIs are marked as "deprecated". This means that the API was defined in an earlier version of the specification, but is no longer recommended for use. In most cases, a preferred alternative API is available. Implementations are permitted to continue providing the deprecated API, and applications are encouraged to convert from the deprecated API to the recommended alternative API. Deprecated APIs are likely to be removed completely in a subsequent version.

1.6.1 Parameter Range Notation

Valid parameter ranges are specified using both enumerated lists of valid values and sequential ranges. The ranges are specified using the following interval notation [**Error! Reference source not found.**]: (a, b) for open intervals, [a, b] for closed intervals, and (a, b] and [a, b) for half-closed intervals, defined as follows:

$$(a,b) = \{x \mid a < x < b\}$$
$$[a,b] = \{x \mid a \le x \le b\}$$
$$(a,b] = \{x \mid a < x \le b\}$$
$$[a,b) = \{x \mid a \le x < b\}$$

1.6.2 Format and Typographic Conventions

This document uses the following conventions:

Table 2: Format and Typographic Conventions

Format	Meaning
`Courier font`	Sample code, API parameter specifications

1.7 Acknowledgements

The OpenMAX AL specification is the result of the contributions of many people. The following is a partial list of contributors in order of company name then contributers surname, including the respective companies represented at the time of their contribution:

Stewart Chao, AMD (now with Qualcomm)
Wilson Kwan, AMD (now with Qualcomm)
Chris Grigg, Beatnik
Andrew Ezekiel Rostaing, Beatnik
Tim Granger, Broadcom
Roger Nixon, Broadcom
Wolfgang Schildbach, Coding Technologies
Nathan Charles, Creative
Lewis Balfour, Ericsson
Harald Gustafsson, Ericsson
Håkan Gårdrup, Ericsson
Glenn Kasten, Google
Jean-Michel Trivi, Google
Brian Murray, Freescale
Erik Noreke, Independent
Yeshwant Muthusamy, Nokia (Chair)
Matti Paavola, Nokia (Past Chair)
Scott Peterson, NVIDIA
Isaac Richards, NVIDIA
Neil Trevett, NVIDIA
Jim Van Welzen, NVIDIA (Past Chair)
Tom Longo, Qualcomm (Past Editor)
John Mortimer, QSound
Mark Williams, QSound
Ytai Ben-Tsvi, Samsung
Natan Linder, Samsung
Gal Sivan, Samsung
David Eaton, SRS Labs
Jefferson Hobbs, SRS Labs
Robert Alm, ST-Ericsson
Joakim Roubert, ST-Ericsson
Weijun Jiang, SKY MobileMedia
Stephan Tassart, ST Microelectronics
Brian Evans, Symbian
James Ingram, Symbian
Leo Estevez, Texas Instruments
Danny Jochelson, Texas Instruments

2 OpenMAX AL Features and Profiles

OpenMAX AL is designed with media application developers in mind. It provides support for a number of audio, video and image features that facilitate the development of a wide range of applications on the target devices. Supported features include:

- **Media playback:** Includes playback of PCM audio, encoded audio, MIDI ringtones, UI sounds, encoded video and image content as well as extraction of metadata embedded in the media content. Video playback refers to support for synchronized audio/video playback. Image playback refers to the decoding and display of compressed image data.
- **Media recording:** Includes support for recording of audio and video, and image capture. Video recording refers to support for synchronized audio/video recording. Image capture refers to camera functionality.
- **Effects and controls:** For audio, includes support for general controls such as volume and balance, and music player effects such as equalizer. For image and video, includes support for controlling the brightness, contrast and gamma adjustments.

Optional functionality includes:

- **MIDI:** Includes support for SP-MIDI, mobile DLS, and mobile XMF.
- **Analog Radio:** Includes support for analog radio tuning as well as support for RDS/RBDS content.
- **LED array:** Includes support controlling multiple colored LED arrays.
- **Vibration device ("vibra"):** Includes support for controlling vibration device intensity and frequency.

Section 2.4 discusses optional features in the API.

These features enable the development of multimedia-rich applications such as media players, media recorders, and games.

2.1 Motivation

The definition of OpenMAX AL profiles accommodates the fact that OpenMAX AL may be used for a range of devices catering to different market segments. Not all implementations of OpenMAX AL will need (or can accommodate) all of the functionality represented by this large set of features. For example, a media playback-only device would have little use for implementing all the media recording functionality that is also part of the API.

Thus OpenMAX AL segments the APIs into two groupings of functional affinity relative to typical devices - specifically, a collection of player functionality and a collection of recorder functionality.

2.2 OpenMAX AL Profile Definition

An OpenMAX AL profile is a defined subset of features of the same functional type collectively required on any implementation that claims to support that profile.

2.3 Profiles

OpenMAX AL defines two profiles: **Media Player** and **Media Player/Recorder.** A short description and rationale of each of the profiles is discussed below:

- **Media Player:** This profile encapsulates media playback functionality including the ability to render audio, video and image data in one or more formats. The Media Player profile is appropriate for playback-only devices which do not include any support for capturing or recording media. Personal media players are good examples of devices that would use this profile.
- **Media Player/Recorder:** This profile encapsulates all-inclusive media playback and recording functionality including the ability to capture as well as render audio, video and image data in one or more formats. High-end mobile phones are good examples of devices that would use this profile. This profile subsumes the Media Player profile.

The following table lists the features in the two profiles of OpenMAX AL. A "Y" in a cell indicates that the corresponding API feature is mandatory in that profile, while a blank cell indicates an absence of that feature.

Table 3: Features of the OpenMAX AL Profiles

API (Profile) Feature	Media Player	Media Player/ Recorder
KEY USE CASES		
Playback of audio and video files	Y	Y
Rendering of image sources	Y	Y
Recording and storage of audio and video sources		Y
Capture and storage of image sources		Y
DATA ACCESS		
Support for various media container formats	Y	Y
Specify a stream source (local/remote file, memory/flash, etc.)	Y	Y
Identify data sources by name, such as URL/URI, or by file handle	Y	Y
Respect DRM	Y	Y
Select an input source from among a multitude of available inputs		Y
Select an output destination from among a multitude of available outputs	Y	Y
DEVICE CAMERA		
Camera flash activation		Y
Camera Effects		Y
Exposure settings (exposure time, aperture and ISO sensitivity)		Y
Focus control (including macro-focus on/off)		Y
White balance control		Y
Optical and digital zoom		Y
PLAYBACK, RECORDING AND PROCESSING CONTROLS		
"VCR-type" playback modes: play, pause, stop, rewind, fast-forward	Y	Y
Play multiple sounds at a given time	Y	Y
Playback of raw PCM audio	Y	Y
Playback of sampled audio encoded in a form other than raw PCM	Y	Y

API (Profile) Feature	Media Player	Media Player/ Recorder
Playback of mono and stereo sampled audio	Y	Y
Volume control	Y	Y
Audio balance control	Y	Y
Audio pan control	Y	Y
End-to-end looping of audio/video content	Y	Y
Audio/video segment looping	Y	Y
Seeking to a seek point (such as chapter)	Y	Y
Route media to multiple simultaneous outputs	Y	Y
Set a sound's priority	Y	Y
Audio equalization	Y	Y
Audio recording from a microphone or on-device line-in jack		Y
Audio recording from another software component		Y
Record audio to a non-PCM format		Y
Recording modes: record and stop		Y
Query the estimated size of the output image based on current image settings		Y
PER-APPLICATION SETTINGS		
Use key-value pairs to query and set both the codec and non-codec configurations of the underlying media engine	Y	Y
PER-OBJECT SETTINGS		
Set video encoder properties: frame rate, bitrate (constant/variable), size, resolution, duration limit and codec format		Y
Set audio encoder properties: bitrate (constant/variable), channel count, duration limit, sampling frequency, codec format, size, and resolution		Y
Set image encoder properties: codec format, size and resolution		Y
Set and query image/video encoder special effects, if supported. Effects include, but are not limited to, "monochrome", "sepia", "emboss", "paintbrush", "solarize", "red-eye reduction", "cartoon" and "negative"		Y
METADATA		
Extract metadata from media files and embedded media streams	Y	Y
Insert/edit metadata in recorded media content		Y
EVENT AND ERROR NOTIFICATIONS		
Callbacks for periodic media positioning (such as for progress bar)	Y	Y
Callback alerts when playback is in a prescribed position (such as for looping)	Y	Y
Callbacks for error conditions	Y	Y
CAPABILITY QUERIES		

API (Profile) Feature	Media Player	Media Player/ Recorder
Enumerate and query the capabilities of available input sources		Y
Enumerate and query the capabilities of available output destinations	Y	Y
Query the API version number	Y	Y
Query capabilities of the OpenMAX AL implementation	Y	Y
MISCELLANEOUS		
Query the degree to which an OpenMAX AL implementation is based on OpenMAX IL: none, partial or full	Y	Y
Extensibility	Y	Y
Minimum 16-bit PCM audio output	Y	Y

2.4 Optionality Rules of Features and Profiles

In an effort to minimize confusion among developers and reduce fragmentation, OpenMAX AL adheres to the following rules on features and profiles:

1. All features within a profile are mandatory – this is critical in assuring developers and implementers that if they pick a profile, all the functionality representative of that profile will indeed be present. Then, applications written towards a specific profile will indeed be portable across OpenMAX AL implementations of that profile.
2. A feature that does not fit in any of the profiles is considered to be an optional feature. OpenMAX AL does not categorize optional features in any way, to avoid a potentially confusing combinatorial explosion and effectively negating the benefits of the first rule. Vendors are free to pick and choose from the entire set of optional features to augment their implementations of either of the two profiles. An important exception to this rule is MIDI functionality. See section 2.5 for an explanation of the special designation for MIDI in OpenMAX AL.
3. Vendors are free to implement features from more than one profile, but they can claim compliance with a profile only if they implement all of the features of that profile. Example: If a vendor implements the Media Player profile in its entirety and adds just audio recording from the recorder profile, then that vendor can only claim compliance with the Media Player profile.

The following table lists some of the optional features in OpenMAX AL with the rationale for their optionality.

Table 4: Optional Features in OpenMAX AL

Optional Feature	Reason for Optionality
Reception and playback of analog radio content	Implies dependency on radio hardware (such as FM radio chip).
Controlling the analog radio tuner of the device	Implies dependency on radio hardware
Retrieve RDS (Radio Data System) content from the currently tuned station.	Implies dependency on radio hardware.
MIDI functionality	See Section 2.5 for the special designation for MIDI in OpenMAX AL

2.5 MIDI in OpenMAX AL

MIDI is considered fundamental functionality for mobile phones, an important class of target devices for OpenMAX AL. However, there exist other target devices for AL such as personal media players and recorders that typically have no use for MIDI functionality. Mandating MIDI in OpenMAX AL would place a gratuitous burden on such devices to implement MIDI just to claim compliance with OpenMAX AL. For this reason, MIDI is designated as optional functionality. All MIDI-related features are in an optional "+ MIDI" category (an exception to the second rule defined in section 2.4). Implementations that do support MIDI functionality can lay claim to this "+ MIDI" designation in addition to the profile(s) they support. Example: "AL Media Player + MIDI" or "AL Media Player/Recorder + MIDI". However, to avoid fragmentation of the API with respect to MIDI, an implementation can lay claim to the "+ MIDI" designation only if it supports all of the following MIDI-related features:

- MIDI file playback and related callbacks
- SP-MIDI
- Mobile DLS
- Mobile XMF

2.6 Profile Notes

Profiles notes are used within this specification to identify objects and interfaces where support is optional in one or more of the profiles. Objects and interfaces without profile notes are mandated in all profiles. Here are some representative examples of profile notes found in the specification:

PROFILE NOTES
This object represents an optional feature and consequently optional in all profiles.

PROFILE NOTES
This interface is mandated only in the Media Player profile.

2.7 Behavior for Unsupported Features

If an application attempts to use a feature that is not present in a specific implementation of OpenMAX AL, the implementation shall fail the request to use the feature, returning the `XA_RESULT_FEATURE_UNSUPPORTED` error code (see section 3.4 Error Reporting). This can happen either when calling `GetInterface()` on an unsupported interface or when attempting to call an unsupported method in an interface. Furthermore, if an interface with an unknown ID is used (either during object creation or in a `GetInterface()` call), this same result code shall be returned. This facilitates portability of applications using non-standard extensions.

3 Design Overview

3.1 Object Model

3.1.1 Objects and Interfaces

The OpenMAX AL API adopts an object-oriented approach using the C programming language. The API includes two fundamental concepts on which are built all other constructs: an object and an interface.

An **object** is an abstraction of a set of resources, assigned for a well-defined set of tasks, and the state of these resources. An object has a type determined on its creation. The object type determines the set of tasks that the object can perform. This can be considered similar to a class in C++.

An **interface** is an abstraction of a set of related features that a certain object provides. An interface includes a set of **methods**, which are functions of the interface. An interface also has a type which determines the exact set of methods of the interface. We can define the interface itself as a combination of its type and the object to which it is related.

An **interface ID** indentifies an interface type. This identifier is used within the source code to refer to the interface type.

Objects and interfaces are tightly related – an object **exposes** one or more interfaces, all of which have different interface types, that is, an object may contain at most one interface of each interface type. A given interface instance is **exposed** by exactly one object. The application controls the object's state and performs the object operations exclusively through the interfaces it exposes. Thus, the object itself is a completely abstract notion, and has no actual representation in code, yet it is very important to understand that behind every interface stands an object.

The relationship between object types and interface types is as follows. The object type determines the types of interfaces that may be exposed by objects of this type. Each object type definition in this document includes a detailed list of the interfaces on that object.

PROFILE NOTES

The set of interface types allowed for a given object type may vary across profiles. This will be explicitly stated in this document, per object, and per interface type.

An object's **lifetime** is the time between the object's **creation** and its **destruction**. The application explicitly performs both object creation and destruction, as will be explained later in this document.

An object maintains a state machine with the following states:

- **Unrealized (initial state):** The object is alive but has not allocated resources. It is not yet usable; its interfaces' methods cannot yet be called.
- **Realized:** The object's resources are allocated and the object is usable.
- **Suspended (optional state):** The object has fewer resources than it requires in order to be usable but it maintains the state it was in at the moment of suspension. This state is optional to the extent that, in the face of resource loss, the implementation has the option of putting an object either in the Suspended state or the Unrealized state.

The following state diagram illustrates the states and transitions.

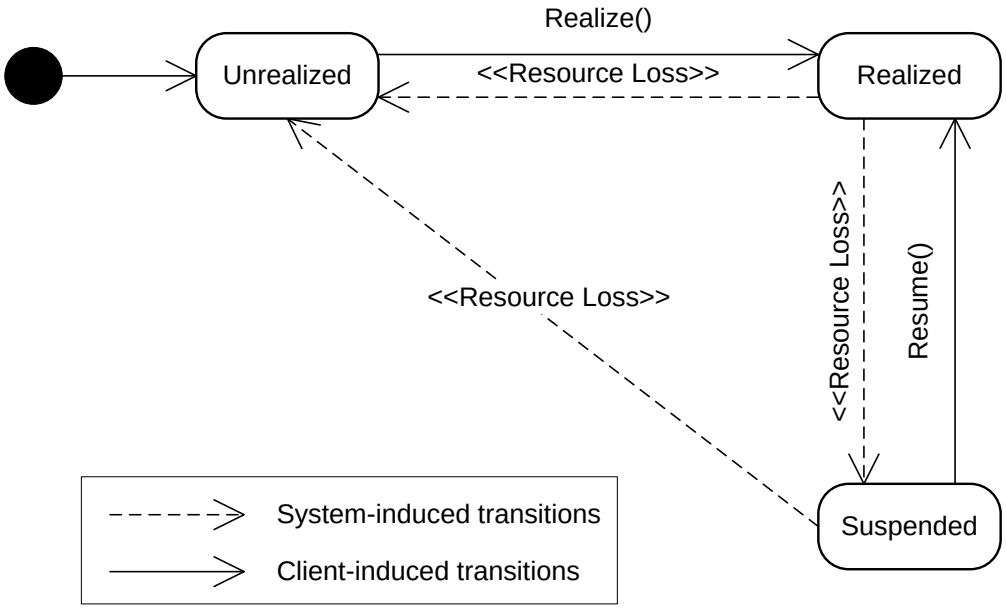


Figure 4: Object state diagram

When the application destroys an object, that object implicitly transitions through the Unrealized state. Thus it frees its resources and makes them available to other objects.

See section 3.1.7 for more details on resource allocation.

3.1.2 Getters and Setters

Getters and setters provide access to object properties. An application uses a setter to change the value of an object's property and a getter to retrieve a value of an object's property.

Unless explicitly specified in a getter's method name, a getter shall always return the exact value that was set by a previous call to a setter, even if that value had been rounded off internally by the implementation. An exception to this rule is that a Boolean getter must return only logically (but not necessarily numerically) the same value as was set.

Here is a short example that demonstrates the use of a getter and a setter:

```
XAresult res;
XAVolumeItf volumeItf;
XAmillibel vol;
.
.
.
res = (*volumeItf)->GetVolumeLevel(volumeItf, &vol); CheckErr(res);
res = (*volumeItf)->SetVolumeLevel(volumeItf, vol + 7); CheckErr(res);
```

Unless specified otherwise, applications are responsible for allocation and deallocation of memory buffers used in the interface methods.

3.1.3 Representation in Code

As stated in the previous section, objects have no representation in code. OpenMAX AL refers an object via its `XAObjectItf` interface (see section 8.24).

The API represents interfaces as C structures, where all the fields are method-pointers, representing the methods. These interface structures are always stored and passed as pointer-to-pointer-to-struct and never by value (this level of indirection enables more efficient API implementations).

Each of the interface methods has a first argument, called `self`, whose type is the interface type. Thus when calling an interface method, the caller must pass the interface pointer in this argument. Additionally, each of the callback prototypes has a first argument called `caller`, whose type is the interace type.

Here is an example of a simple interface:

```
struct XASomeInterfaceItf_;
typedef const struct XASomeInterfaceItf_ * const * XASomeInterfaceItf;

struct XASomeInterfaceItf_ {
    XAresult (*Method1) (
        XASomeInterfaceItf self,
        XAuint32 value
    );
    XAresult (*Method2) (
        XASomeInterfaceItf self,
        XAuint32 * pValue
    );
};
```

This interface is called `XASomeInterfaceItf` and has two methods: `Method1()` and `Method2()`. Such an interface will be used as follows:

```
XAuint32 i;
XAresult res;

XASomeInterfaceItf someItf;

// ... obtain this interface somehow ...
res = (*someItf)->Method1(someItf, 42); CheckErr(res);
res = (*someItf)->Method2(someItf, &i); CheckErr(res);
```

Note that this code excludes the mechanism for obtaining the interface itself, which will be explained in the following sections.

3.1.4 The XAObjectItf Interface

`XAObjectItf` is a special interface type fundamental to the architecture. Every object type in the API **exposes this interface**. It is the "entry-point" for an object since every method that **creates** a new object actually returns the `XAObjectItf` of this object. Using this interface, an application may perform all the basic operations on the object and may access other interfaces exposed by the object.

The application **destroys** the object by calling the `Destroy()` method of the `XAObjectItf` interface.

The application **obtains other interfaces** of the object by calling the `GetInterface()` method of the `XAObjectItf` interface. The application retrieves an interface by its type ID which uniquely identifies it; an object cannot contain more than one interface of a certain type.

The application **controls the state** of the object by calling the `Realize()` and `Resume()` methods of the `XAObjectItf` interface.

For a complete and detailed specification of this interface type, see section 8.24.

3.1.5 The Engine Object and XAEngineItf Interface

Other fundamental entities in the architecture are the engine object and the `XAEngineItf` interface. These entities serve as the API's entry-point. The application begins an OpenMAX AL session by creating an engine object.

The engine object is created using the global function `xaCreateEngine()`
(see section 6.1). The result of engine object creation is a `XAObjectItf` interface, regardless of the implementation.

An implementation can optionally support multiple versions of the API. When the application creates the engine object, it specifies the version of the engine object. If the requested version of the API is supported, an engine object of that version is returned. Any object or interface created or derived from that engine object will adhere to the same version of the specification as the engine object.

After creating the engine object, the application can obtain this object's `XAEngineItf` interface. This interface contains creation methods for all the other object types in the API.

To create an object process:

- Create and realize an engine object if one has not been created already.
- Obtain the `XAEngineItf` interface from this object.
- Call the appropriate object creation method of this interface.
- If the call succeeds, the method will return an `XAObjectItf` interface of the new object.
- After working with the object, call the `Destroy()` method of the `XAObjectItf` to free the object and its resources.

For a complete and detailed specification of the engine object and the `XAEngineItf` interface type, please refer to sections 7.2 and 8.11 respectively.

3.1.6 The Relationship Between Objects and Interfaces

The set of interfaces exposed by an object is determined by three factors:

- The object's type
- The interfaces requested by the application during the object's creation
- The interfaces added and removed by the application during the object's lifetime

An object's type determines the set of interfaces that will always exist, regardless of whether the application requests them or not. These interfaces are called *implicit interfaces* and every object type has its own set of implicit interfaces that will exist on every object of this type. The `XAObjectItf` interface, introduced in section 3.1.4, is fundamentally required on every object, so it is designated as implicit on all object types, that is, the application never needs to explicitly request that it be exposed on any object.

Every object type also defines a set of interfaces that are available on objects of this type, but will not be exposed by an object unless explicitly requested by the application during the object's creation. These explicitly requested interfaces are called *explicit interfaces.*

Finally, every object type also defines a set of interfaces that may be added and removed by the application during the object's lifetime. These types of interfaces are called *dynamic interfaces* and they are managed by a dedicated interface, called `XADynamicInterfaceManagementItf` (see section 8.10), which enables this dynamic binding. Attempts to dynamically add or remove implicit interfaces on an object will fail.

The set of explicit and dynamic interfaces for a given object type may vary between implementations (see section 3.5). However, for a given profile, each object type has a set of mandated explicit interfaces and a set of mandated dynamic interfaces that shall be supported by every implementation.

When an application requests explicit interfaces during object creation, it can flag any interface as required. If an implementation is unable to satify the request for an interface that is not flagged as required, this will not cause the object to fail creation. On the other hand, if the interface is flagged as required and the implementation is unable to satisfy the request for the interface, the object will not be created.

The following table summarizes whether an object is created and an interface is exposed, according to how the specification relates their types and how the application designates the interface at the object's creation.

Table 5: Interface Exposure Rules During Object Creation

			Determined by the application		
			Interface requested by application		**Interface not requested by application**
			Interface marked as required	**Interface not marked as required**	
Determined by implementation & specification	**Mandated interface**	**Implicit**	✓	✓	✓
		Explicit	✓	✓	✗
	Optional interface	**Available**	✓	✓	✗
		Not available	💣	✗	✗

Key:

✓	Object is created and interface is exposed, subject to resource constraints
✗	Object is created but interface is not exposed
💣	Object is not created and interface is not exposed

The next table summarizes whether interface is exposed on an object when the application requests to add the interface dynamically, according to whether the specification mandates support for the interface on the object type and whether this support is mandated dynamically or not.

Table 6: Interface Exposure Rules for Dynamic Adding of Interfaces

			Determined by application
			Application dynamically adds interface
Determined by implementation &	**Mandated**	**Mandated interface**	✓

	dynamic	**Optional interface**	**Available**	✓
			Not available	✘
	Not mandated dynamic			?

Key:

✓	Interface is exposed, subject to resource constraints
✘	Interface is not exposed
?	Interface maybe exposed (implementation dependant), subject to resource constraints

3.1.7 The XADynamicInterfaceManagementItf Interface

The `XADynamicInterfaceManagementItf` interface provides methods for handling interface exposure on an object after the creation and realization of the object. The `XADynamicInterfaceManagementItf` itself is an implicit interface on all object types.

The dynamic nature of an interface is unrelated to it being optional or mandated for an object. An interface that is "mandated" as dynamic can actually be realized both at object creation time as well as dynamically, at any point in the object's lifetime (using the `XADynamicInterfaceManagementItf`). Interfaces that represent a significant resource drain if realized on object creation but that are never used, are prime candidates for dynamic interfaces. By making them dynamic, the application developer can use them only when needed, often resulting in significant resource optimization. Dynamic interfaces are explicitly called out in the "Mandated Interfaces" sections of the corresponding objects in section 7.

Although this interface provides a method for requesting the acquisition of an additional interface, namely `AddInterface()`, the implementation may deny the request. The criteria for denial are implementation dependent. For example, the state of an object's player or recorder may influence the success or failure of dynamic interface addition. Upon a successful `AddInterface()` call for a specified interface, that interface is immediately usable. There is no separate call to "realize" the given interface. The interface instance is obtained, just as for static interfaces, by using the `GetInterface()` method.

An application may retire a dynamic interface with a `RemoveInterface()` call. After a `RemoveInterface()` call, the dynamic interface is no longer usable. When an object is unrealized, all interfaces, including the dynamic interfaces, are unusable and effectively removed from the object.

3.1.8 Resource Allocation

The exact amount of resources available on an OpenMAX AL implementation may vary across different implementations and over the lifetime of an engine object. As a result, an application using the OpenMAX AL API must always be prepared to handle cases of failure in object realization or dynamic interface addition. In addition, an object's resources may be stolen by another entity (such as another object or the underlying system) without prior notice.

To allow applications to influence the resource allocation by the system, a priority mechanism is introduced. Priority values are set on a per-object basis. Applications can use these object priorities to influence the behavior of the system when resource conflicts arise. For example, when a high-priority object needs a certain resource and the resource is currently assigned to a lower-priority object, the resource will most likely be "stolen" from the low-priority object (by the system) and re-assigned to the high-priority object. An application can change the priority of

an object at any point during the lifetime of the object. It is also worth noting that these object priorities set by the application are confined to this instance of the API engine. They are unrelated to the priorities that may be assigned by the system to the application itself and other components running on the system, for the purposes of resource management.

When a resource is stolen from an object, this object will automatically transition to either the Suspended state or the Unrealized state, depending on whether its interface states are preserved or reset, respectively. To which of the states the object transitions is determined solely by the implementation. When in either of these two states, all of this object's interfaces, except for the `XAObjectItf` interface and the `XADynamicInterfaceManagementItf` interface, become unusable, and return an appropriate error code. Dynamic interfaces are treated the same as any other interfaces. If the object the dynamic interface is exposed on is Suspended or Unrealized, the dynamic interfaces will be suspended or unrealized, respectively.

The application may request to be notified of such a transition. This is done by registering for a notification on the object. The application may also request to be notified when resources become available again, which may allow for the object to regain usability. The notification will include any dynamic interfaces as well, that is, the notification is sent when all the interfaces and the object can have their resources. Individual notification is NOT sent for each dynamic interface.

The application may attempt to recover an Unrealized or Suspended object by calling its `Realize()` or `Resume()` methods, respectively. If the call succeeds, the object will return to the Realized state, and its interface states will be either recovered or reset to default, depending on whether it was unrealized or suspended. The `RemoveInterface()` method is special and can be used in any object state to retire dynamically exposed interfaces. This may help in successfully realizing or resuming the object.

When a stolen resource is freed, the implementation checks whether this resource can be used in order to recover an interface in a resources stolen state. The check is made in object priority order, from high to low. It is not guaranteed, however, that attempting to recover an object after getting this notification will succeed.

An important difference regarding interfaces that are exposed dynamically is how resources are managed. When a dynamic interface loses its resources, a notification is sent but the object state is not affected. Also, other interfaces on the same object are not affected. The application may register for notification of dynamic interface resource changes.

After a lost resources notification, the dynamically exposed interface will become unusable. Two different types of lost resources notification can be received– resource lost, and resource lost permanently. The first type of notifications indicates that the dynamic interface may be resumed by the application after a resource available notification has been received. When the `ResumeInterface()` call succeeds, the dynamic interface will be fully recovered. The second type of notification means that the current instance of the exposed dynamic interface can't recover from the resource loss and shall be retired by the application.

3.2 Threading Model

3.2.1 Mode of Operation

The OpenMAX AL API is generally synchronous. This means that an API method will return only after its operation is complete, and any state changes caused by the method call will be immediately reflected by subsequent calls.

However, in some specific cases, a synchronous operation is not desirable, due to operations that may take a long time. In such cases, the actual termination of the operation will be signaled by a notification. Any state changes caused by the call are undefined between the time of the call and until the time of notification.

Asynchronous functions will be clearly designated as such in their documentation. Otherwise, a synchronous mode of operation should be assumed.

3.2.2 Thread Safety

The OpenMAX AL API may operate in one of two modes, which determine the behavior of the entire API regarding reentrancy:

- **Thread-safe mode:** The application may call the API functions from several contexts concurrently. The entire API will be thread-safe – that is, any combination of the API functions may be invoked concurrently (including invocation of the same method more than once concurrently) by multiple application threads, and are guaranteed to behave as specified.
- **Non-thread-safe mode:** The application needs to take care of synchronization and ensure that at any given time a maximum of one API method is being called. The entire API is not thread-safe – that is, the application needs to ensure that at any given time a maximum of one of the API functions is being executed, or else undefined behavior should be expected.

An implementation shall support one or more of these modes.

The mode of operation is determined on engine creation, and cannot be changed during the lifetime of the engine object. An implementation shall support at least one of these modes, and should document which modes are supported.

Note that an application written to work with non-thread-safe mode will be able to work with a thread-safe mode engine without change. As a result, a valid implementation of thread-safe mode is automatically considered a valid implementation of the non-thread-safe mode; however, implementations of both modes may choose to implement them differently. Implementers should note that implementation of thread-safe mode assumes knowledge of the threading mechanisms used by the application.

3.3 Notifications

In several cases, the application needs to be notified when some event occurred inside the OpenMAX AL implementation, such as when playback of a file has ended, or when an asynchronous method has completed. These notifications are implemented as callback functions – the application registers a method whose signature is specified by the API, and this method will be called by the OpenMAX AL implementation every time a certain event occurs.

Callback functions are registered per-interface and per-event type, thus registering a callback for a certain event on a given object (through one of its interfaces) will not cause this callback to be called if the same event occurs on a different object, or if a different event occurs on the same object. The event type is simply designated by the method that was used to register the callback.

At any given time, a maximum of one callback function may be registered per-interface, per-event type. Registering a new callback on the same interface, using the same registration method, will un-register the old callback. Similarly, registering `NULL` is the way to un-register an existing callback without registering a new one.

The context from which the callbacks are invoked is undefined, and typically implementation- and OS-dependant. Thus the application cannot rely on any system call or OpenMAX AL API call to work from within this call. However, to avoid a dead-end, each implementation should document the list of functions that can be safely called from the callback context. It is highly recommended that the implementation provide at least the means of posting

messages to other application threads, where the event shall be handled. In addition, the `XAThreadSyncItf` interface (see section 8.35) must be usable from within the callback context.

The application should be aware of the fact that callbacks may be invoked concurrently with other callbacks, concurrently with application invocation of an API method, or even from within API calls, and thus should be prepared to handle the required synchronization, typically using the `XAThreadSyncItf` interface (see section 8.35).

For more specific details, refer to the engine object documentation in section 7.2.

3.4 Error Reporting

Almost every API method indicates its success or failure by a result code (except for methods that are not allowed to fail under any circumstances). An API method's documentation states the valid result codes for that method and an implementation shall return one of these result codes. For synchronous methods, the result code is the return value of the method. For asynchronous functions, the result code is contained in the data of the notification sent upon the completion of the operation.

Every API method has a set of pre-conditions associated with it, consisting of:

- API state in which the method should be called
- Context from which the method can be called

The pre-conditions are clearly documented for every method. When the application violates any of the pre-conditions, the method call will fail, and the method's result code will indicate the violation. The API will remain stable and its state will not be affected by the call. However, it is recommended that applications do not rely on this behavior and avoid violating the pre-conditions. The main value of this behavior is to aid in the debug process of applications, and to guarantee stability in extreme conditions, and specifically under race-conditions.

However, the API's behavior may be undefined (and even unstable) in any of the following conditions:

- Corruption of the `self` parameter, which is passed as every method's first parameter, or any other parameter passed by pointer.
- Violation of the threading policy.

3.5 Memory Management

3.5.1 General

The application is responsible for allocating and deallocating all memory originating within the application space. OpenMAX AL is responsible for allocating and deallocating all memory originating within the OpenMAX AL implementation. At no point shall the application deallocate memory allocated by the OpenMAX AL implementation, and the OpenMAX AL implementation will not deallocate memory allocated by the application. Exceptions to this rule are not allowed.

3.5.1.1 Parameters

Parameters passed to OpenMAX AL methods are used during the duration of the method call. The application is free to change or deallocate any parameters or parameter associated memory after a call to OpenMAX AL, with the exception of memory buffers. Memory buffers passed from the application to the OpenMAX AL implementation are to be kept valid by the application for the duration of the use by the

OpenMAX AL implementation. The duration of the memory buffer use and when the application can deallocate the memory is described in the interface or object description where they are used.

3.5.2 Callbacks

The information passed to the application from the OpenMAX AL implementation in a callback is valid for the duration of the callback only. The OpenMAX AL implementation is free to deallocate any memory associated with a callback to the application as soon as the callback returns. The application is responsible for saving any information passed to it in a callback required for later use. This includes any memory buffers passed from the OpenMAX AL implementation to the application.

3.5.3 Exceptions

Any exceptions and behavior in contrast to the memory management rules stated above will be described in the interface description which behaves in contrast to the above rules.

3.6 Extensibility

3.6.1 Principles

The OpenMAX AL API was designed with extensibility in mind. An extended API is defined as one that provides functionality additional to that defined by the specification, yet considered still conforming to the specification.

The main principles of the extensibility mechanism are:

- Any application written to work with the standard API will still work, unchanged, on the extended API.
- For an application that makes use of extensions, it will be possible and simple to identify cases where these extensions are not supported, and thus to degrade its functionality gracefully.

Possible extensions may include vendor-specific extensions as well as future versions of OpenMAX AL.

3.6.2 Permitted Modifications to Physical Code

The OpenMAX AL header files shall be edited only for the following purpose:

- To amend definitions of types (for example, 32 bit signed integers) such that they have correct representation.

Any vendor-specific extensions to the API shall reside in header files other than the OpenMAX AL header files.

3.6.3 Extending Supported Interface Types

An extended API may introduce new interface types and expose these interfaces on either existing object types or on extended object types (see section 3.6.4).

An extended API may also expose standard interfaces on standard / extended object types that do not normally require exposing these interfaces.

The extended interfaces will be defined in a manner similar to standard interfaces. The extended interface types will have unique IDs, generated by the extension provider.

Note that the extending API may not alter standard interfaces or apply different semantics on standard interfaces, even if the syntax is preserved. An exception to this rule is extending the valid parameter range of functions, detailed later.

Functions may not be added to any of the interfaces defined in the specification. To do that, a new interface which includes the desired standard interface must be defined, along with a new interface ID which must be generated,

It is also highly recommended that whenever an interfaces signature changes (even slightly), a new interface ID will be generated, and the modified interface will be considered a new one. This is to protect applications already written to work with the original interface.

3.6.4 Extending Supported Object Types

An extended API may introduce new object types to those specified in the standard API. The extended objects may expose either standard or extended interface types. Should it expose standard interfaces – they must still behave as specified. Otherwise, the extended API may provide extended interface types with different semantics.

The extended objects will be created by extended interfaces with creation functions. These extended interfaces typically will be exposed by the standard engine object, but can also be exposed on other objects.

3.6.5 Extending Method Parameter Ranges

An extended API may support an extended range of parameters for a standard method than the range mandated by the specification. The semantics of the extended range are left to the extended API's specification. However, for mandated ranges, the API shall behave exactly according to the specification.

> Care must be taken when the extended API is vendor-specific in these cases – future versions of the API may use these extended values for different purposes. To help guard against collisions with future API versions, implementations of an extended API shall have the most significant bit set on any extensions to an enumeration (a fixed set of discrete unsigned values). For example:
> ```
> #define XA_SEEKMODE_FAST          ((XAuint32) 0x0001)
> #define XA_SEEKMODE_ACCURATE      ((XAuint32) 0x0002)
> /* ACME extension to SEEKMODE enumeration */
> #define XA_SEEKMODE_ACME_FOO      ((XAuint32) 0x8001)
> ```

The most significant bit does not need to be set for any extensions to continuous ranges or for signed values.

3.6.6 Result Codes

It is not possible to extend the result codes for any standardized method in the API. An implementation shall return one of the result codes listed in the method's documentation.

3.6.7 Data Locators

An extended API may introduce new data locators to those specified in the standard API. The extended data locators may be used in place of the data locators defined in the main specification. The extension will specify which objects accept the data locators defined by the extension.

3.6.8 Interface ID Allocation Scheme

A common interface ID allocation scheme shall be used for vendor-specific interface IDs, to prevent collisions by different vendors.

The UUID mechanism provided freely in the Web-site below is highly recommended to be used by all providers of extensions.

http://www.itu.int/ITU-T/asn1/uuid.html

The interface IDs do not have to be registered – it is assumed that the above mechanism will never generate the same ID twice.

3.6.9 Avoiding Naming Collisions

It is recommended that vendors providing extended APIs qualify all the global identifiers and macros they provide with some unique prefix, such as the vendor name. This prefix will come after the API prefix, such as `XAAcmeDistortionItf`.

This is meant to reduce the risk of collisions between vendor-specific identifiers and other versions of the specification of other vendors.

The example below demonstrates using extensible features of the specification. The code will compile both on implementations which support the extended API as well as those which do not:

```
void ShowExtensibility(XAEngineItf *eng)
{
    XAresult res;
    XAboolean supported;
    XAObjectItf player;
    XAAcmeDistortionItf distortionItf;
    XAPlayItf playbackItf;
    XAmillibel vol;

    /* create an audio player */
    res = eng->CreateMediaPlayer(eng, &player, ...); CheckErr(res);
    res = (*player)->GetInterface(player,
        &XA_IID_ACME_DISTORTION, (void*)&distortionItf);
    if (XA_RESULT_FEATURE_UNSUPPORTED == res)
    {
        supported = false;
    }
    else
    {
        CheckErr(res);
        supported = true;
    }

    /* continue using the player normally whether
       the extension is supported or not */
    res = (*player)->GetInterface(player, &XA_IID_PLAYBACK,
                                  (void*)&playbackItf);
    CheckErr(res);

    ...
```

```
    ...

    /* whenever calling an extension's method,
       wrap it with a condition. */
    if (supported)
    {
        /* employ one of the interface's methods */
        res = (*distortionItf)->SetDistortionGain(distortionItf, vol);
        CheckErr(res);
    }
}
```

4 Functional Overview

4.1 Object Overview

OpenMAX AL represents entities in its architecture as objects, including:

- Engine Object
- Media Objects
- Metadata Extractor Object
- Audio Output Mix Objects
- Camera Objects
- LED Array Objects
- Radio Objects
- Vibration Control Objects

The following sections provide an overview of each of these.

4.1.1 Engine Object

The engine object is the entry point to the OpenMAX AL API. This object enables you to create all the other objects used in OpenMAX AL.

The engine object is special in the sense that it is created using a global function, `xaCreateEngine()` (see section 6.1). The result of the creation process is the `XAObjectItf` interface (see section 8.24) of the engine object. The implementation is not required to support the creation of more than one engine at a given time.

The engine object can have two different modes of operation, thread-safe mode and non-thread safe mode. The application specifies the mode of operation upon engine object creation. See section 3.2 for details.

The engine object shall expose the `XAThreadSyncItf` interface (see section 8.35) to enable synchronization between the API's callback contexts and the application contexts.

After creation of the engine object, most of the work will be done with the `XAEngineItf` interface (see section8.13) exposed by this object.

An additional functionality of the engine object is querying implementation-specific capabilities. This includes the encoder and decoder capabilities of the system. The OpenMAX AL API gives implementations some freedom regarding their capabilities, and these capabilities may even change according to runtime conditions. For this reason, during runtime the application may query the actual capabilities. However, this specification defines a minimum set of capabilities, expressed as a set of use-cases that shall be supported on every implementation, according to the profiles that are implemented. These use-cases are described in detail in section 4.7.

The engine object represents the system's various multimedia-related devices via unique device IDs. It supports the enumeration of audio input, audio output, camera, radio, LED and vibrator devices as well as mechanisms to query their capabilities. Applications can use information regarding the devices' capabilities to:

- Determine if they can even run on the system (for example, an application that can render only 8 kHz 8-bit audio might not be able to run on a system that can handle only sampling rates of 16 kHz and above at its outputs.)

- Configure the user interface accordingly so that the user is presented with the correct device choices in the UI menus.

The audio I/O device capabilities interface is described in section 8.3.

4.1.2 Media Objects

A media object implements a multimedia use case by performing some media processing task given a prescribed set of inputs and outputs. Media objects include (but are not limited to) objects that present and capture media streams, often referred to as *players* and *recorders*, respectively. They operate on audio, video, and image data or some combination of them.

The following characteristics define a media object:

- The operation it performs, denoted by the creation method used to instantiate the media object.
- The inputs it draws data from, denoted as its *data sources* and specified at media object creation.
- The outputs it sends data to, denoted as its *data sinks* and specified at media object creation.

The media object creation methods are described in section 8.13.

4.1.2.1 Data Source and Sink Structures

A data source is an input parameter to a media object specifying from where the media object will receive a particular type of data (such as audio, video, or image). A data sink is an input parameter to a media object specifying to where the media object will send a particular type of data (such as audio, video, or image).

The number and types of data sources and sinks differ from one media object to another. The following characteristics define a data source or sink:

- Its *data locator* which identifies where the data resides. Possible locators include:
    - URIs (such as a filename)
    - Memory addresses
    - I/O devices
    - Output mixes
    - Cameras
    - Content pipes
- Its *data format* which identifies the characteristics of the data stream. Possible formats include:
    - MIME-type based formats
    - PCM formats
    - RAW image formats

An application specifies a media object's respective data source(s) and sink(s) upfront in the creation method for the media object. Collectively, the media object together with its associated source(s) and sinks(s) define the use case the application wants executed.

4.1.3 Metadata Extractor Object

Player objects support reading of the metadata of the media content and recorder objects support writing metadata. However, sometimes it is useful just to be able to read metadata without having to be able to playback the media. A Metadata Extractor object can be used for reading metadata without allocating resources for media playback. Using this object is recommended particularly when the application is interested only in presenting metadata without playing the content and when wanting to present metadata of multiple files. The latter is particularly interesting for generating playlists for presentation purposes because a player object would unnecessarily allocate playback resources. Furthermore, players cannot change their data source dynamically; therefore, for metadata extraction from multiple files, the application needs to create and destroy player objects many times, which is both inefficient, and may result in fragmentation of the heap. A Metadata Extractor object does not have a data sink, but it has one data source that can be dynamically changed.

4.1.4 Audio Output Mix Object

The API allows for routing of audio to multiple audio outputs and includes an audio output mix object that facilitates this functionality. The application retrieves an output mix object from the engine and may specify that output mix as the sink for a media object. The audio output mix object is specified as a sink for a media object using the `XA_DATALOCATOR_OUTPUTMIX` data locator as described in section 9.2.25. The engine populates the output mix with a set of default audio output devices. The application may query for this list of devices or request changes to it via the `XAOutputMixItf` interface. The API does not provide a direct audio output IO-device as a sink for media objects.

The audio output mix object is defined in section 7.7 and the output mix interface is described in section 8.25.

4.1.5 Camera Object

Control of one of the device's cameras is handled via the Camera object. The number of cameras supported by a device and their capabilities can be retrieved from the media engine via the `XACameraCapabilitiesItf` interface. A camera I/O device exposes the `XACameraItf` interface, which is used to control camera features such as flash, focusing, metering, exposure compensation, sensitivity, shutter speed, aperture, white balance, and zoom. The camera I/O object is connected to a media recorder media object to capture still images or video.

4.1.6 LED Array Control Object

Control of the device's LEDs is handled via the LED array object. Querying the capabilities of and creating a LED array object is an engine-level operation, while control over individual LEDs is handled by the object.

4.1.7 Radio Object

Control of the device's FM/AM radio is handled via the Radio object. Instantiating a Radio object is an engine-level operation. The Radio I/O device can expose two interfaces: `XARadioItf` and `XARDSItf`. `XARadioItf` is an implicit interface that is used to control the basic functionality, such as setting the frequency. `XARDSItf` is an optional interface to control Radio Data System functionality.

The Radio I/O device object shall be connected to a Media Player object to make the radio audible.

4.1.8 Vibration Control Object

Control of the device's vibration support is handled via the Vibra object. Querying the capabilities of and creating a Vibra object is an engine-level operation, while control of the actual vibration is handled by the object.

4.2 Audio Playback and Recording

This section introduces OpenMAX AL functionality for the playback and recording of sampled audio content.

An audio player is used for sampled audio playback. OpenMAX AL supports both file-based and in-memory data sources, as well as piped content. The API supports data encoded in many formats, although the formats supported by a device are implementation-dependent.

An audio recorder is used for capturing audio data. Audio capture is an optional component of OpenMAX AL for implementations of the Media Player profile.

4.3 Video Playback and Recording

This section introduces OpenMAX AL functionality for the playback and recording of video.

A video player supports the playback of synchronized audio/video content and video content absent of audio. OpenMAX AL supports both file-based and in-memory data sources as well as piped content. The API supports data encoded in many formats, although the formats supported by a device are implementation-dependent. An application can also use a video player to display the preview window for a camera.

A video recorder is used for capturing synchronized audio/video content and video content absent of audio. Video capture is an optional component of OpenMAX AL for implementation of the Media Player profile.

4.4 Image Renderering and Capture

This section introduces OpenMAX AL functionality for the rendering and capture of image content.

An image player supports the rendering of image data. OpenMAX AL supports both file-based and in-memory data sources as well as piped content. The API supports data encoded in many formats, although the formats supported by a device are implementation-dependent.

An image recorder supports the capture of image data. Captured snapshots may be encoded or raw. Image capture is an optional component of OpenMAX AL for implementations of the Media Player profile.

4.5 Playback of MIDI

OpenMAX AL supports MIDI playback using the standard player creation mechanism, the creation method `XAEngineItf::CreateMediaPlayer()`. This method provides the ability to specify a MIDI data source and an audio output device, as well as an optional data source for an instrument bank data source and data sinks for an LED array output device and a Vibra output device. OpenMAX AL supports MIDI data sources that refer to SP-MIDI [SP-MIDI] and Mobile XMF [mXMF] files. Playback is controlled via the standard OpenMAX AL interfaces, such as `XAVolumeItf`, `XAPlaybackItf`, `XAPlaybackRateItf`, and `XASeekItf`. MIDI players also support metadata extraction via the `XAMetadataExtractionItf`.

4.5.1 Support for Mobile DLS

OpenMAX AL supports Mobile DLS [mDLS] soundbanks as stand-alone files provided to a media player object on creation or embedded within a Mobile XMF file. In addition, the media player supports the GM soundbanks [MIDI] by default.

In several cases, a media player will not be able to handle two DLS banks at the same time (for example, bank provided during media player creation and bank embedded in the content). In such a case, player creation may fail, and the application can retry the creation without providing the additional bank.

When a program is selected for a MIDI channel (using bank select / program change messages), the media player will first look for the program in the embedded DLS bank, if such exists. If it is not found, the media player will look in the DLS bank that was provided on creation, if applicable. If it is still not found, the media player will try to load the program from the built-in GM bank. If the program does not exist there either, the media player shall generate silence on the specified channel, but should still keep track of that channel's state.

4.6 Display Regions

Conceptually OpenMAX AL maintains three separate notions relating to the visual output:

- **Display:** Corresponds to the entire logical screen area (or "desktop"). The display is provided by the native windowing manager in the form of a native handle. All interaction with this entity is handled exclusively via native interfaces using the native handle (not via OpenMAX AL). Multiple media objects may operate on the same display.
- **Window:** Corresponds to the window within the display. A window may be sized to encompass the entire display and yet it is considered an independent entity (for instance, a fullsize window may be later sized down to a region less than the entire display or have other windows on top of or behind it). The window is provided by the native windowing manager in the form of a native handle. All interaction with this entity is handled exclusively via native interfaces using the native handle (not via the OpenMAX AL). Multiple media objects may operate on the same window.
- **Region:** Corresponds to the area within the native window where the media object presents its output. This may include the entire window area or be some subset of the window area. The region is specific to a particular media object.

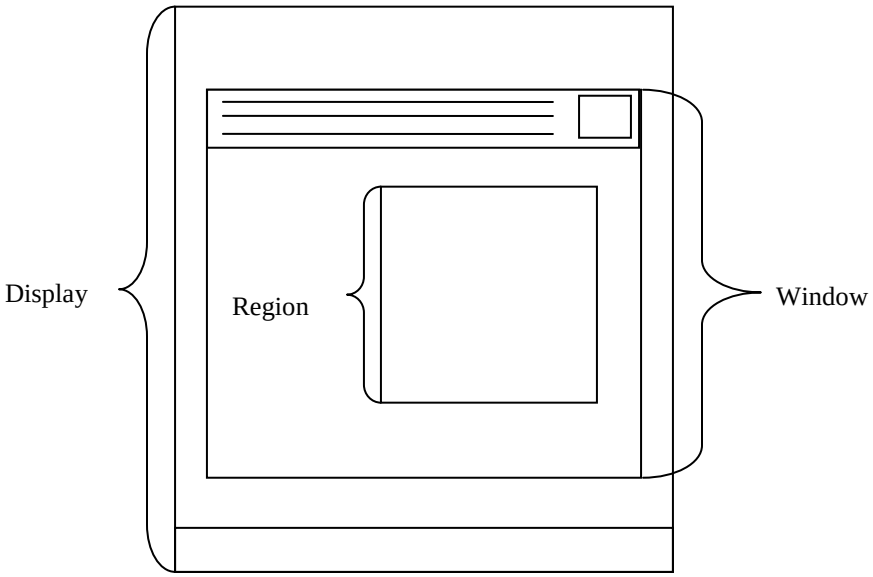


Figure 5: Display with windowed rendering

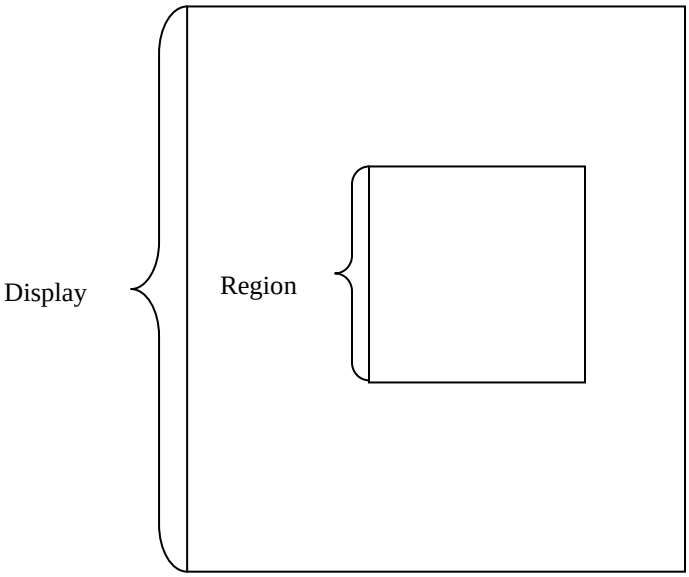


Figure 6: Display with full-screen rendering

The native display handle and the native window handle associated with an AL media object are of type `XA_DATALOCATOR_NATIVEDISPLAY` and reside in the data sink structure. Together these handles provide media objects the hooks necessary to interact with the native windowing manager. Note that OpenMAX AL does not presently standardize the interaction with the native windowing manager but assumes this communication will take place via platform specific interfaces.

4.7 OpenMAX AL Use Cases

This section illustrates the typical use of objects and interfaces in some typical cases of OpenMAX AL use. The support for these use cases is mandatory in all profiles unless otherwise stated in profile notes. We indicate optional interfaces and objects with grey color.

4.7.1 Audio and Video Playback

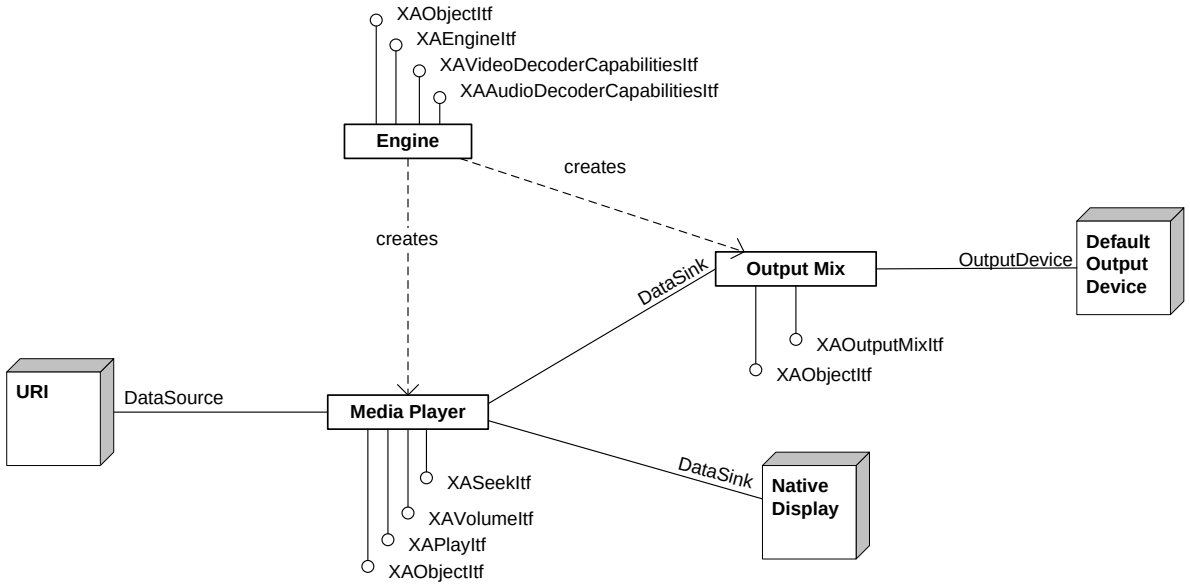


Figure 7: Audio and video use case

The Media Player object facilitates audio and video playback. A Media Player is created using the `XAEngineItf` interface of the engine object. Upon creation, we associate the Media Player with an Output Mix (which we create via the `XAEngineItf` interface) for audio output and with a native display handle for video output. We also set the data source of the Media Player during creation. The data source could be, for example, a URI pointing to a video file in the local file system. The Output Mix is by default associated with the system-dependent default output device.

PROFILE NOTES

The support for this use case is mandated in all profiles.

4.7.2 Audio Playback

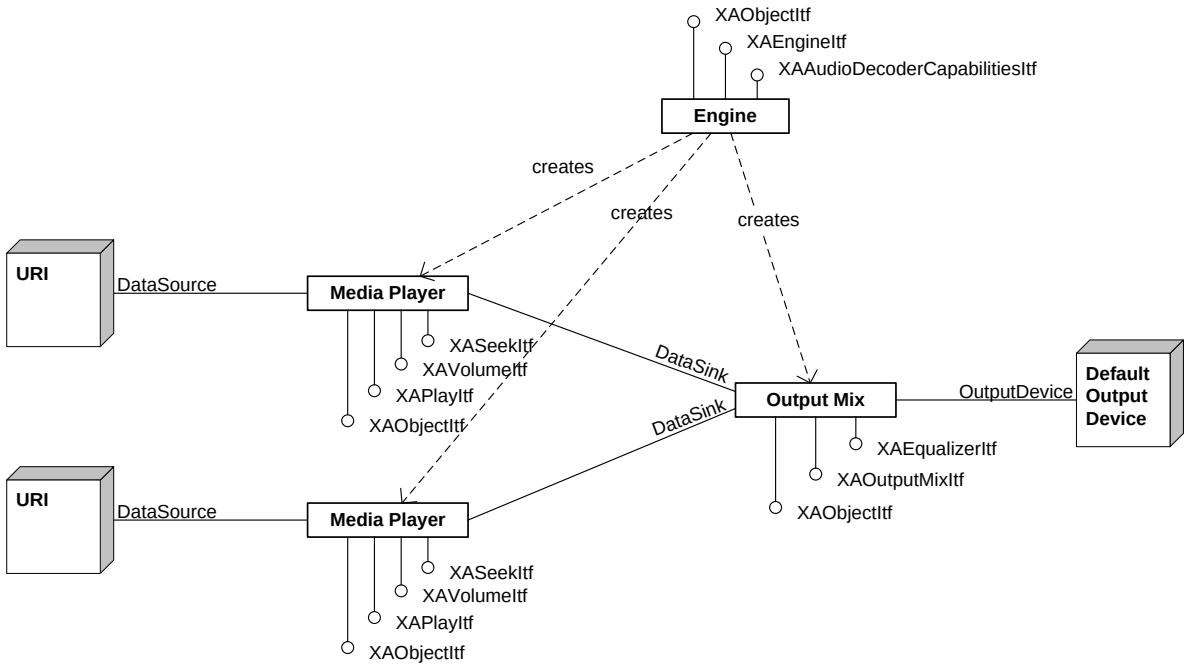


Figure 8: Audio playback with multiple players use case

OpenMAX AL may support playback multiple audio files simultaneously. This use case leverages two Media Player objects for audio playback. We create the Media Players using the `XAEngineItf` interface of the engine object. Upon creation, we associate the Media Players with an Output Mix (which we created with the `XAEngineItf` interface) for audio output. We also set the data sources of the Media Players during creation. The data sources can be, for example, URIs pointing to audio files in the local file system. The Output Mix is by default associated with the system-dependent default output device.

PROFILE NOTES

The support for this use case is mandated in all profiles.

4.7.3 Recording Audio

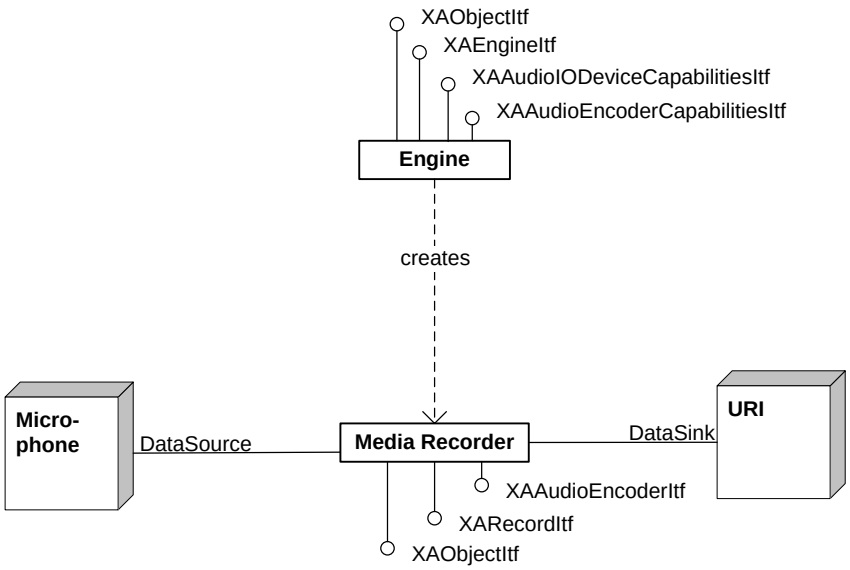


Figure 9: Recording audio use case

An Audio Recording use case is handled by an Media Recorder object. It is created using `XAEngineItf` interface of the engine object. Upon creation, it is associated with an audio data source, which can be, for example, a microphone (an audio input device). The data sink of the Media Recorder can be a URI pointing to an audio file in the local file system on which the audio will be recorded.

PROFILE NOTES

The support for this use case is mandated only in the Media Player/Recorder profile.

4.7.4 Image Player

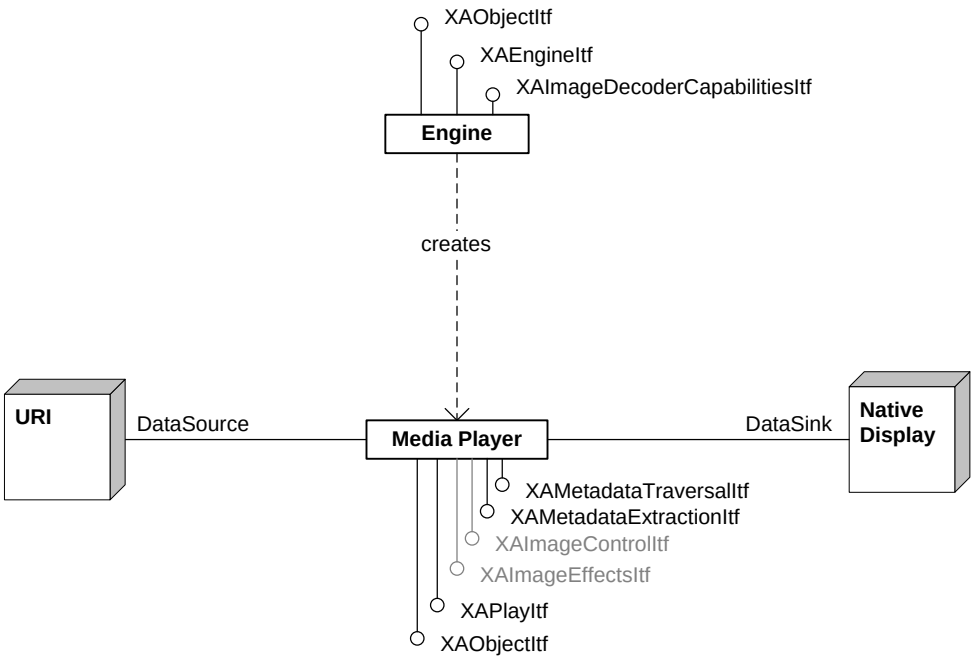


Figure 10: Image player use case

Media Player object also supports image playback. We create a Media Player using the `XAEngineItf` interface of the engine object. Upon creation, we associate the Media Player with a native display handle for image output. We also set the data source of the Media Player during creation. The data source can be, for example, a URI pointing to an image file in the local file system.

PROFILE NOTES

The support for this use case is mandated in all profiles.

4.7.5 Video Camera

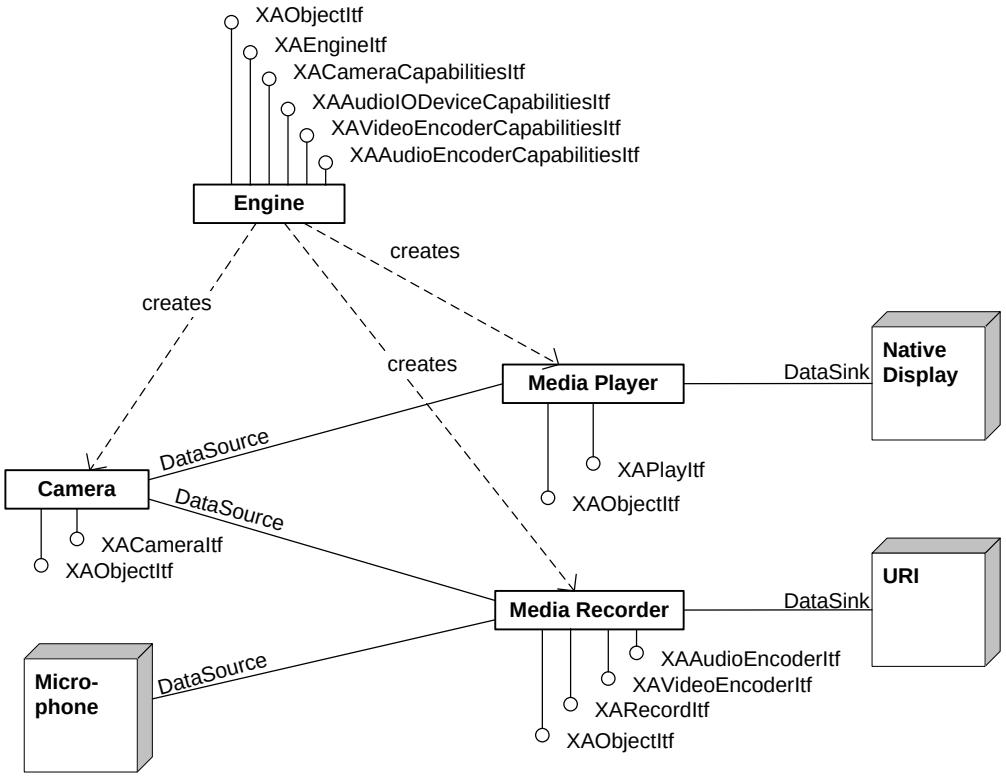


Figure 11: Video camera use case

Video camcorder use case requires a Media Recorder object for recording and a Media Player for the viewfinder. We create both using the `XAEngineItf` interface of the engine object. Upon creation, we associated both with the same Camera object (which we create with the `XAEngineItf` interface). We set the audio data source of the Media Recorder to be a microphone (an audio input device). The data sink for the Media Player is a native window or display handle (as it was in the previous video playback use case). The data sink of the Media Recorder can be a URI pointing to a video file in the local file system where the data will be recorded.

PROFILE NOTES

The support for this use case is mandated only in the Media Player/Recorder profile.

4.7.6 Still Camera

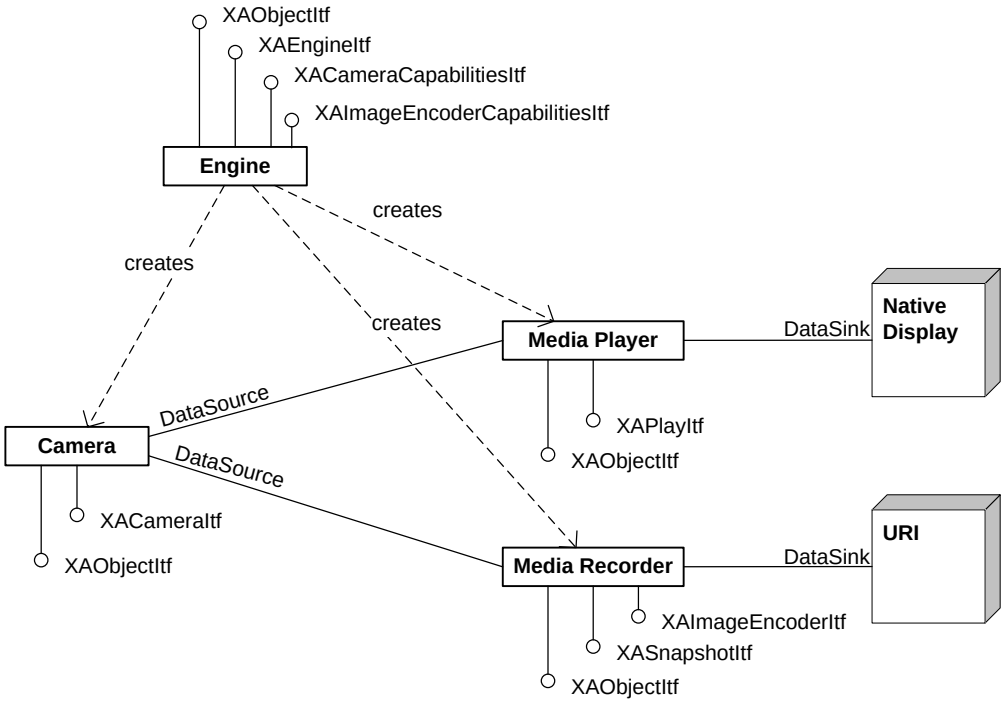


Figure 12: Still camera use case

Still camera use case is similar to the video camera use case except the Media Recorder exposes different interfaces. It provides the `XASnapshotItf` interface for still image capture and `XAImageEncoderItf` for the image encoder settings (instead of the `XARecordItf` and `XAVideoEncoderItf` interfaces respectively)

PROFILE NOTES

The support for this use case is mandated only in the Media Player/Recorder profile.

4.7.7 Radio Playback

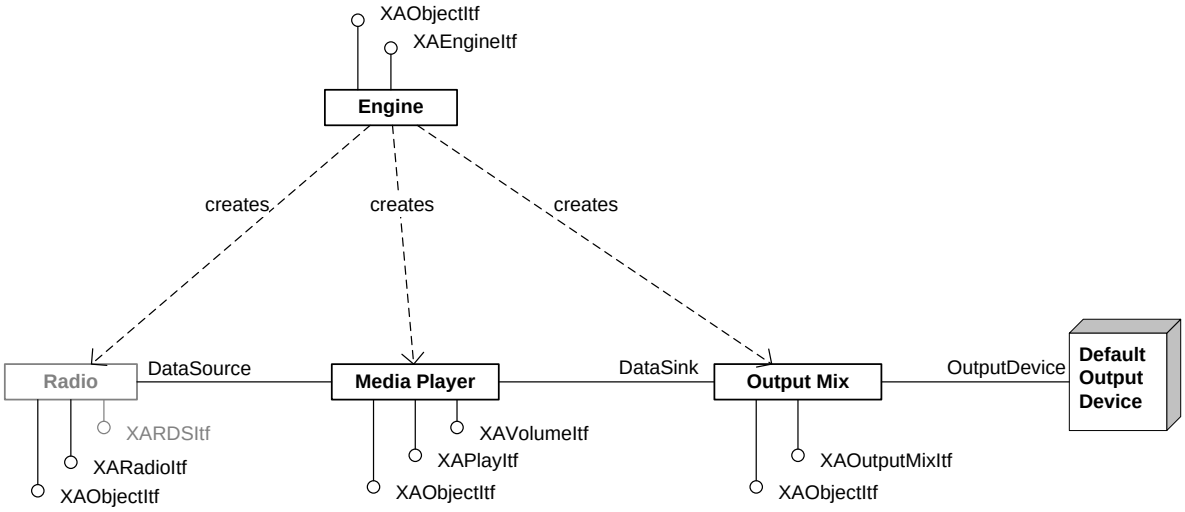


Figure 13: Radio playback use case

A Media Player object may also facilitate the radio playback use case. As always, we create the Media Player using the `XAEngineItf` interface of the engine object. Upon creation, we associate the Media Player with an Output Mix (which we create with the `XAEngineItf` interface) for audio output. By default, OpenMAX AL automatically associates the Output Mix with the system-dependent default output device. During the creation, we set the Radio I/O device (which we create with the `XAEngineItf` interface) as the data source.

PROFILE NOTES

The support for this use case is optional in all profiles since support for Radio I/O device object is optional.

4.7.8 Reading Metadata

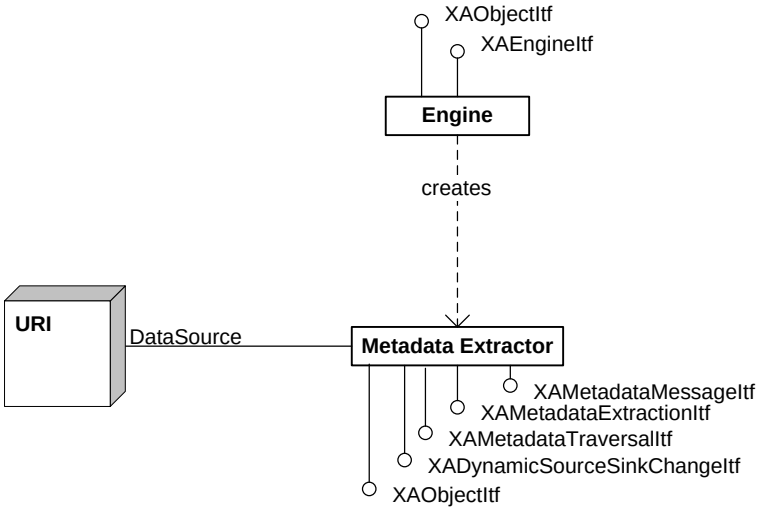


Figure 14: Reading meta data use case

A Metadata Extractor object will read the metadata of a media file without allocating resources for audio playback. As in other use cases, we create the object using `XAEngineItf` interface of the engine object and, upon creation, we set the data source of the Metadata Extractor. The data source is typically a URI pointing to a media file in the local file system. However, the Metadata Extractor supports the `XADynamicSourceSinkChangeItf` interface which we can use to change the data source. Therefore we may extract metadata from multiple files (in series) without creating a new Metadata Extractor object for every single file. The `XAMetadataExtractionItf` and `XAMetadataTraversalItf` interfaces are used for actually reading and traversing the metadata from a file. The XAMetadataMessageItf is used to set callbacks that execute whenever a metadata item is encountered.

PROFILE NOTES

The support for this use case is mandated in all profiles.

PART 2: API Reference

5 Base Types and Units

OpenMAX AL defines a set of cross-platform fixed width types that are used within the API. The definition of these are system-dependent and the platform provider must specify these types. OpenMAX AL also defines a set of types for different units required by the API, such as distance and volume. To aide programmability, most of these units are based on the thousandth unit of a SI unit [ISO1000].

5.1 Standard Units

The table below shows the standard types for units used in OpenMAX AL.

Table 7: OpenMAX AL Unit Types

Unit	Measurement	C type
Angle	millidegree (mdeg)	`XAmillidegree`
Distance	millimeter (mm)	`XAmillimeter`
Frequency	milliHertz (mHz)	`XAmilliHertz`
Scale/Factor	permille (‰)	`XApermille`
Time	millisecond (ms)	`XAmillisecond`
Time	microsecond (µs)	`XAmicrosecond`
Volume level	millibel (mB)	`XAmillibel`

5.2 Base Types

```
typedef <system dependent> XAint8;
typedef <system dependent> XAuint8;
typedef <system dependent> XAint16;
typedef <system dependent> XAuint16;
typedef <system dependent> XAint32;
typedef <system dependent> XAuint32;
typedef <system dependent> XAuint64;
typedef <system dependent> XAfloat32;
typedef <system dependent> XAfloat64;
typedef <system dependent> XAchar;
typedef XAuint32           XAboolean;
typedef XAint16            XAmillibel;
typedef XAuint32           XAmillisecond;
typedef XAuint32           XAmilliHertz;
typedef XAint32            XAmillimeter;
typedef XAint32            XAmillidegree;
typedef XAint16            XApermille;
typedef XAuint32           XAmicrosecond;
typedef XAuint64           XAtime;
typedef XAuint32           XAresult;
```

Type	Description
XAint8	An 8-bit signed type. The definition of this type is system-dependent.
XAuint8	An 8-bit unsigned type. The definition of this type is system-dependent.
XAint16	A 16-bit signed type. The definition of this type is system-dependent.
XAuint16	A 16-bit unsigned type. The definition of this type is system-dependent.
XAint32	A 32-bit signed type. The definition of this type is system-dependent.
XAuint32	A 32-bit unsigned type. The definition of this type is system-dependent.
XAuint64	A 64-bit unsigned type. The definition of this type is system-dependent.
XAfloat32	A 32-bit floating point type. The definition of this type is system-dependent.
XAfloat64	A 64-bit floating point type. The definition of this type is system-dependent.
XAchar	A character type. All strings within the API, except where explicitly defined otherwise, are UTF-8, null-terminated, XAchar arrays. The definition of this type is system-dependent.
XAboolean	A Boolean type, where zero is false and all remaining values are true.
XAmillibel	A type for representing volume in millibels (mB), one thousandth of a Bel, one hundredth of a decibel.
XAmillisecond	A type for representing time in milliseconds, (ms), one thousandth of a second).
XAmilliHertz	A type for representing frequency in milliHertz (mHz), one thousandth of a Hertz.
XAmillimeter	A type for representing distance in millimetres (mm), one thousandth of a meter.
XAmillidegree	A type for representing an angle in millidegrees (mdeg), one thousandth of a degree.
XApermille	A type for representing a scale or factor in permille. One permille (1‰) is equal to a factor of 0.001. One thousand permille (1000‰) is equal to a factor of one.
XAmicrosecond	A type for representing time in microseconds, one millionth of a second).
XAtime	A type for representing time measured as seconds since midnight, 1st January 1970 UTC.
XAresult	A type for standard OpenMAX AL errors that all functions defined in the API return.

6 Functions

6.1 xaCreateEngine

xaCreateEngine			

`XA_API XAresult XAAPIENTRY xaCreateEngine(`
`  XAObjectItf * pEngine,`
`  XAuint32 numOptions,`
`  const XAEngineOption * pEngineOptions,`
`  XAuint32 numInterfaces,`
`  const XAInterfaceID * pInterfaceIds,`
`  const XAboolean * pInterfaceRequired`
`)`			

Description	Initializes the engine object and gives the user a handle.		
Pre-conditions	None		
Parameters	`pEngine`	[out]	Pointer to the resulting engine object.
	`numOptions`	[in]	The number of elements in the options array. This parameter value is ignored if `pEngineOptions` is `NULL`. If `numOptions` is equal to 0, the engine is initialized without any optional features.
	`pEngineOptions`	[in]	Array of optional configuration data. A `NULL` value initializes the engine without the optional features being enabled.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
	`pInterfaceIds`	[in]	An array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
	`pInterfaceRequired`	[in]	An array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`
	`XA_RESULT_FEATURE_UNSUPPORTED`
	`XA_RESULT_RESOURCE_ERROR`
	`XA_RESULT_ENGINEOPTION_UNSUPPORTED`		

xaCreateEngine	
Comments	The options supported by an individual implementation are implementation-dependent. Standardized options are documented in section.9.2.32. The engine is destroyed via the destroy method in `XAObjectItf`. See Appendix D: for examples using this method.
See Also	Engine object [see section 7.2].

6.2 xaQueryNumSupportedEngineInterfaces

xaQueryNumSupportedEngineInterfaces			
XA_API XAresult XAAPIENTRY xaQueryNumSupportedEngineInterfaces(**XAuint32 * pNumSupportedInterfaces** **);**			
Description	Queries the number of supported interfaces available on engine object.		
Parameters	`pNumSupportedInterfaces`	[out]	Identifies the number of supported interfaces available. Must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The number of supported interfaces will include both mandated and optional interfaces available for the engine object.		
See also	`xaQuerySupportedEngineInterfaces()`, `XAEngineItf::QueryNumSupportedInterfaces` [see section 8.13].		

6.3 xaQuerySupportedEngineInterfaces

xaQuerySupportedEngineInterfaces			
XA_API XAresult XAAPIENTRY xaQuerySupportedEngineInterfaces(**XAuint32 index,** **XAInterfaceID * pInterfaceId** **);**			
Description	Queries the supported interfaces on engine object.		
Pre-conditions	None		
Parameters	`index`	[in]	Incrementing index used to enumerate available interfaces. Supported index range is 0 to N-1, where N is the number of supported interfaces.
	`pInterfaceId`	[out]	Identifies the supported interface corresponding to the given `index`. Must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The number of supported interfaces will include both mandated and optional interfaces available for the engine object.		
See also	`xaQueryNumSupportedEngineInterfaces()`, `XAEngineItf::QueryNumSupportedInterfaces` [see section 8.13].		

7 Object Definitions

This section documents all the object types supported by the API. Some object types are mandated to be supported only in a selection of the profiles. Where this is the case, the object's description will include a profile note stating this.

Each object type has a list of **mandated interfaces** that must be supported for that object type. If the object type itself is not mandated, if the implementation allows creation of objects of that type, it must still support all the mandated interfaces for the object type. The list of mandated interfaces may vary according to profile, as documented in the profiles notes. The mandated interface sections also documents whether an interface is implicit or must be supported dynamically.

Besides of the mandated interfaces, an object is free to support any interfaces defined in this specification (and any vendor-specific interfaces). However, some interfaces specified in this specification make much more sense with a specific object type than some other interfaces. Therefore, for information only, each object type has also a list of *applicable optional interfaces*. The implementer is not limited to support only these listed interfaces, but these lists provide the application developer a hint concerning which optional interfaces might be supported.

7.1 Camera I/O Device

Description

The camera I/O device object provides access to still image and video data from a camera source and exposes controls for camera-related settings. A Media Recorder object may leverage a camera object to capture image or video data. See section D.5 for an example using this object.

PROFILE NOTES
Creation of objects of this type is mandated in the Media Player/Recorder profile.

Mandated Interfaces

XACameraItf [see section 8.6]

This interface controls the camera.

This interface is an implicit interface on this object.

XAObjectItf [see section 8.24]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding interfaces to the object after creation.

This interface is an implicit interface on this object.

Applicable Optional Interfaces

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

XAImageControlsItf [see section 8.15]

This interface exposes controls for brightness, contrast and gamma adjustments.

XAImageEffectsItf [see section 8.17]

This interface controls the image effects.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAVideoPostProcessingItf [see section 8.41]

This interface controls scaling, mirroring, cropping and rotating.

7.2 Engine Object

Description

This object type is the entry point of the API. An implementation shall enable creation of at least one such object, but attempting to create more instances (either by a single application or by several different applications) may fail.

The engine object supports creation of all other OpenMAX AL objects via its `XAEngineItf` interface, and querying of the implementation's capabilities via interfaces. See Appendix D: for examples using this object.

PROFILE NOTES
Creation of objects of this type is mandated in all profiles.

Creation

An engine is created using the global function `xaCreateEngine()` (see section 6.1).

Mandated Interfaces

XAEngineItf [see section 8.13]

This interface exposes methods for creation of other OpenMAX AL objects.

This interface is an implicit interface on this object.

XAObjectItf [see section 8.25]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding interfaces to the object after creation.

This interface is an implicit interface on this object.

XAAudioIODeviceCapabilitiesItf [see section 8.3]

This interface exposes methods for querying available audio device capabilities.

This interface is an implicit interface on this object.

XAAudioDecoderCapabitiesItf [see section 8.2]

This interface exposes methods for querying audio decoder capabilities.

This interface is a mandated interface on this object.

XAAudioEncoderCapabitiesItf [see section 8.4]

This interface exposes methods for querying audio encoder capabilities.

This interface is a mandated interface on this object.

XACameraCapabilitiesItf [see section 8.7]

This interface exposes methods for querying camera device capabilities.

This interface is mandated only for Media Player/Recorder profile.

XAImageDecoderCapabitiesItf [see section 8.16]

This interface exposes methods for querying image decoder capabilities.

This interface is a mandated interface on this object.

XAImageEncoderCapabitiesItf [see section 8.19]

This interface exposes methods for querying image encoder capabilities.

This interface is a mandated interface on this object.

XAVideoDecoderCapabitiesItf [see section 8.38]

This interface exposes methods for querying video decoder capabilities.

This interface is a mandated interface on this object.

XAVideoEncoderCapabitiesItf [see section 8.40]

This interface exposes methods for querying video encoder capabilities.

This interface is a mandated interface on this object.

XAThreadSyncItf [see section 8.36]

This interface exposes methods for entering and exiting critical section.

This interface is a mandated interface on this object.

Applicable Optional Interfaces

XADeviceVolumeItf [see section 8.9]

This interface controls audio input and output device specific volumes.

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

7.3 LED Array I/O Device

Description

The LED array I/O device object encapsulates and controls a set of LEDs. Its functionality covers setting LED color, activating and deactiving LEDs.

PROFILE NOTES
This object is a standardized extension and consequently optional in all profiles.

Mandated Interfaces

XAObjectItf [see section 8.25]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XALEDArrayItf [see section 8.20]

This interface exposes all LED capabilities for a LED array `IODevice`.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding dynamic interfaces (see section 3.1.6) to the object.

This interface is an implicit interface on this object.

Applicable Optional Interfaces

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

7.4 Media Player Object

Description

The media player object plays audio, video or image content as specified by the data source. It performs any implicit decoding, applies any specified processing and performs synchronized rendering of final audio, video, and image streams to the destinations specified by the audio and image/video data sinks.

The application may omit the audio sink or image/video sink when creating a media player (that is, by passing `NULL` as a parameter) if the media player does not wish to playback the desired content.

Media players may render data from a variety of sources. The controls exposed in the playback interface are appropriate for all sources. Though conceptually identical, the precise effect of each playback state depends on the use case. Unlike the playback interface, the seek and playback rate controls may be inappropriate for sources where such functionality is not achievable. For instance, persistent time-based content is amenable to seeking and rate control, yet live data is amenable to neither.

To illustrate, consider these typical use cases:

The playback of an audio-only, video-only or audio-and-video file. The ability to seek and set rate on this player depends on the nature of the content and/or the capabilities of the media player implementation. The media player performs any synchronization inherent in audio-and-video content. In the context of this use case, the playback states have the following interpretation:

Table 8: Presentation and Playback State for Audio/Video Media

Playback State	Audio Content	Image/Video Content	Presentation
Stopped	not presented	not presented	not updating
Playing	presented	presented	updating
Paused	not presented	presented	not updating

The display of an image file. Due to its instantaneous nature, an image file cannot be seeked and its rate cannot be changed. In the context of this use case, the playback states have the following interpretation:

Table 9: Presentation and Playback State for Image Media

Playback State	Audio Content	Image/Video Content	Presentation
Stopped	not applicable	not presented	not applicable
Playing	not applicable	presented	not applicable
Paused	same as playing state		

The rendering of live preview data. In this case, the source is a camera (as in the case of a viewfinder). In this context, the playback states have the following interpretation:

Table 10: Presentation and Playback State for Live Preview

Playback State	Audio Content	Image/Video Content	Presentation
Stopped	not applicable	not presented	not updating

Playback State	Audio Content	Image/Video Content	Presentation
Playing	not applicable	presented	updating
Paused	not applicable	presented	not updating

A application may choose to associate both a media player object and a media recorder object with the same camera I/O device object to implement a camcorder or still image camera application. In such a case, if the application instructs the media recorder to freeze on a snapshot, any media player that shares the camera I/O device object will transition to the paused playback state.

See section D.1, section D.2, section D.3 and section D.5 for examples using this object.

PROFILE NOTES
Creation of objects of this type is mandated in all profiles.

Mandated Interfaces

XAPlayItf [see section 8.27]

This interface controls the playback state of the media player.

This interface is an implicit interface on this object.

XAPrefetchStatusItf [see section 8.29]

This interface enables querying the prefetch status of the audio player.

This interface is a mandated interface on this object.

XASeekItf [see section 8.33]

This interface controls the position of the playback head and any looping of playback.

This interface is a mandated interface on this object.

XAMetadataExtractionItf [see section 8.21]

This interface exposes methods for retrieving metadata.

This interface is a mandated interface on this object.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAMetadataMessageItf [see section 8.23]

This interface is used to set metadata callbacks during streaming and/or local playback. To be used in conjunction with XAMetadataExtractionItf.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAMetadataTraversalItf [see section 8.24]

This interface exposes methods for navigating through metadata.

This interface is a mandated interface on this object.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAObjectItf [see section 8.25]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding interfaces to the object after creation.

This interface is an implicit interface on this object.

XAVolumeItf [see section 8.42]

This interface exposes volume-related controls.

This interface is a mandated interface on this object for media containing audio.

XAStreamInformation [see section 8.35]

This interface exposes stream property interface and selection.

This interface is a mandated interface on this object.

Applicable Optional Interfaces

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

XADynamicSourceItf [see section 8.11]

This interface is deprecated. Use `XADynamicSourceSinkChangeItf` [see section 8.12] instead.

This interface enables changing the data source of the player post-creation.

XADynamicSourceSinkChangeItf [see section 8.12]

This interface enables changing the data source or the data sink of the object post-creation.

XAEqualizerItf [see section 8.14]

This interface controls a player-specific equalizer effect.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAImageControlsItf [see section 8.15]

This interface exposes controls for brightness, contrast and gamma adjustments.

XAImageEffectsItf [see section 8.17]

This interface controls the image effects.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAPlaybackRateItf [see section 8.28]

This interface exposes playback rate related controls.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAVideoPostProcessingItf [see section 8.41]

This interface controls scaling, mirroring, cropping and rotating.

7.5 Media Recorder Object

Description

The media recorder records audio, video or image content to the destination specified by the data sink. The media recorder captures it from the inputs specified as data sources and performs any specified encoding or processing.

The application may omit the audio source or image/video source when creating a media recorder (that is, by passing `NULL` as a parameter) if the application does not intend to capture the corresponding data type.

Media recorders may capture data from a variety of sources. For instance, a media recorder may support the capture of a segment of audio and/or video data over time and/or instantaneous image data. A media recorder must support either the record interface or the snapshot interface. A media recorder may support both interfaces.

The capture of a segment of audio and/or video data. This type of capture is controlled via the `XARecordItf` interface. The encoding parameters for audio and video are set using the `XAAudioEncoderItf` and `XAVideoEncoderItf`, respectively.

The capture of a still image. This type of capture is controlled via the `XASnapshotItf` interface. The encoding parameters are set using the `XAImageEncoderItf`.

An application may choose to associate both a media player object and a media recorder object with the same camera I/O device object to implement a camcorder or still image camera application. In such a case, if the application instructs the media recorder to freeze on a snapshot, any media player that shares the camera I/O device object will transition to the paused playback state.

See section D.4 and section D.5 for examples using this object.

PROFILE NOTES
Creation of objects of this type is mandated in the Media Player/Recorder profile.

Mandated Interfaces

XARecordItf [see section 8.32]

This interface controls the recording state of the media recorder, enabling the capture of data over some segment of time.

This interface is a mandated interface when recording audio or video media.

XAAudioEncoderItf [see section 8.3]

This interface controls the parameters of audio encoding.

This interface is a mandated interface when recording audio media.

XAAudioEncoderCapabilitiesItf [see section 8.4]

This interface exposes methods for querying audio encoder capabilities.

This interface is a mandated interface when recording audio media.

XAVideoEncoderItf [see section 8.39]

This interface controls the parameters of video encoding.

This interface is a mandated interface when recording video media.

XASnapshotItf [see section 8.34]

This interface enables the capture of still image data.

This interface is a mandated interface when recording image media.

XAImageEncoderItf [see section 8.18]

This interface controls the parameters of image encoding.

This interface is a mandated interface when recording image media.

XAMetadataInsertionItf [see section 8.22]

This interface exposes methods for adding metadata to media.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAObjectItf [see section 8.25]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding interfaces to the object after creation.

This interface is an implicit interface on this object.

Applicable Optional Interfaces

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

XADynamicSourceItf [see section 8.11]

This interface is deprecated. Use `XADynamicSourceSinkChangeItf` [see section 8.12] instead.

This interface enables changing the data source of the recorder post-creation.

XADynamicSourceSinkChangeItf [see section 8.12]

This interface enables changing the data source or the data sink of the object post-creation.

XAEqualizerItf [see section 8.14]

This interface controls a recorder-specific equalizer effect.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAImageControlsItf [see section 8.15]

This interface exposes controls for brightness, contrast and gamma adjustments.

XAImageEffectsItf [see section 8.17]

This interface controls the image effects.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

XAMetadataExtractionItf [see section 8.21]

This interface exposes methods for retrieving metadata.

XAMetadataMessageItf [see section 8.23]

This interface exposes methods for setting metadata callbacks. To be used in conjunction with XAMetadataExtractionItf.

XAMetadataTraversalItf [see section 8.24]

This interface exposes methods for navigating through metadata.

XAVideoPostProcessingItf [see section 8.41]

This interface controls scaling, mirroring, cropping and rotating.

XAVolumeItf [see section 8.42]

This interface exposes volume-related controls.

7.6 Metadata Extractor Object

Description

This object can be used for reading metadata without allocating resources for media playback. Using this object is recommended particularly when the application is interested only in presenting metadata without playing the content and when it wants to present metadata of multiple files. The latter is particularly interesting for generation of playlists for presentation purposes because a media player would unnecessarily allocate playback resources. Furthermore, players cannot change their data source dynamically; therefore, for metadata extraction from multiple files, the application needs to create and destroy player objects many times, which is both inefficient, and may result in fragmentation of the heap.

PROFILE NOTES
Creation of objects of this type is mandated in all profiles.

Mandated Interfaces

XAObjectItf [see section 8.25]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding dynamic interfaces (see section 3.1.6) to the object.

This interface is an implicit interface on this object.

XADynamicSourceItf [see section 8.11]

This interface is deprecated. Use XADynamicSourceSinkChangeItf [see section8.12] instead. This interface exposes controls for changing the data source during the lifetime of the object, to be able to read metadata from multiple files without creating a new object for every single file.

This interface is an implicit interface on this object.

XADynamicSourceSinkChangeItf [see section 8.12]

This interface enables changing the data source or the data sink of the object post-creation.

This interface is an implicit interface on this object.

XAMetaDataExtractionItf [see section 8.21]

This interface exposes controls for metadata extraction.

This interface is an implicit interface on this object.

XAMetaDataMessageItf [see section 8.23]

This interface is used to set metadata callbacks. To be used in conjunction with XAMetadataExtractionItf.

This interface is an implicit interface on this object.

XAMetaDataTraversalItf [see section 8.24]

This interface exposes controls for metadata traversal.

This interface is an implicit interface on this object.

XAStreamInformation [see section 8.35]

This interface exposes stream property interface and selection.

This interface is a mandated interface on this object.

Applicable Optional Interfaces

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

7.7 Output Mix Object

Description

The output mix object represents a set of audio output devices to which one audio output stream is sent. The application retrieves an output mix object from the engine and may specify that output mix as the sink for a media object. The engine must support at least one output mix, though it may support more. The API does not provide a direct audio output IO-device as a sink for media objects.

An output mix is a logical object; it does not (necessarily) represent a physical mix. Thus the actual implementation of the mixing defined logically by the mix objects and their association with media objects is an implementation detail. The output mix does not represent the system's main mix. Furthermore, a mix object represents the application's contribution to the output; the implementation may mix this contribution with output from other sources.

The engine populates the output mix with the default set of audio output devices. The application may request rerouting of that mix via calls to add and remove devices, but whether those requests are fulfilled is entirely the prerogative of the implementation. Furthermore, the implementation may perform its own rerouting of the output mix. In this case, the implementation makes the application aware of changes to the output mix via a notification.

Manipulation of the output mixes leverages the use of device IDs to specify the device(s) operated on. The engine includes a special ID, called the *default device ID*, which represents a set of one or more devices to which the implementation deems audio output should go by default. Although the application may use the default device ID when manipulating an output mix, only the implementation may alter the physical devices this ID represents. Furthermore, the implementation may change the mapping to physical devices dynamically.

See section D.1 and section D.2 for examples using this object.

PROFILE NOTES
Creation of objects of this type is mandated in all profiles.

Mandated Interfaces

XAObjectItf [see section 8.25]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding dynamic interfaces (see section 3.1.6) to the object.

This interface is an implicit interface on this object.

XAOutputMixItf [see section 8.26]

This interface exposes controls for querying the associated destination output devices.

This interface is an implicit interface on this object.

XAEqualizerItf [see section 8.14]

This interface exposes controls over an equalizer effect.

This interface is a dynamic interface on this object. See section 3.1.6 for details on dynamic interfaces.

This interface is a mandated interface on this object.

XAVolumeItf [see section 8.41]

This interface exposes volume-related controls.

This interface is a mandated interface on this object.

Applicable Optional Interfaces

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

7.8 Radio I/O Device

Description

This object represents an radio tuner and maybe used by media objects as a data source. See section D.3 for an example using this object.

PROFILE NOTES
This object is a standardized extension and consequently optional in all profiles.

Mandated Interfaces

XARadioItf [see section 8.30]

This interface exposes control over the basic tuning-related functionality. This interface is also used for switching the radio on or off.

This interface is an implicit interface on this object.

XAObjectItf [see section 8.25]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding interfaces to the object after creation.

This interface is an implicit interface on this object.

Applicable Optional Interfaces

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

XARDSItf [see section 8.31]

This interface exposes Radio Data System functionality.

7.9 Vibra I/O Device

Description

The Vibra I/O device object controls device vibration. Its functionality is limited to activate / deactivate the vibration function of the device, as well as setting its frequency and intensity, if supported.

PROFILE NOTES
This object is a standardized extension and consequently optional in all profiles.

Mandated Interfaces

XAObjectItf [see section 8.25]

This interface exposes basic object functionality.

This interface is an implicit interface on this object.

XADynamicInterfaceManagementItf [see section 8.10]

This interface is used for adding dynamic interfaces (see section 3.1.6) to the object.

This interface is an implicit interface on this object.

XAVibraItf [see section 8.37]

This interface exposes all vibration functionality for a Vibra I/O Device.

This interface is an implicit interface on this object.

Applicable Optional Interfaces

XAConfigExtensionsItf [see section 8.8]

This interface can be used to get and set parameters for any AL object in a vendor-specific manner.

8 Interface Definitions

This section documents all the interfaces and methods in the API.

Almost all methods generate result codes, whether synchronously or asynchronously. Such methods must return either one of the explicit result codes listed in the method's documentation or one of the following result codes:

- `XA_RESULT_RESOURCE_ERROR`
- `XA_RESULT_RESOURCE_LOST`
- `XA_RESULT_INTERNAL_ERROR`
- `XA_RESULT_UNKNOWN_ERROR`
- `XA_RESULT_OPERATION_ABORTED`
- `XA_RESULT_FEATURE_UNSUPPORTED`

For a full definition of these result codes see section 9.2.67.

8.1 XA_IID_NULL

XA_IID_NULL is a placeholder interface ID for the interface array used when creating an object. XA_RESULT_FEATURE_UNSUPPORTED will be returned if the engine tries to create an object that has XA_IID_NULL as a required interface. Also, using it in XAObject::GetInterface() or XADynamicInterfaceManagement::AddInterface() will always fail.

Interface ID

ec7178ec-e5e1-4432-a3f4-4657e6795210

8.2 XAAudioDecoderCapabilitiesItf

Description

This interface provides methods for querying the audio decoding capabilities of the media engine.

This interface provides a means of enumerating all audio decoders available on an engine where a decoderId represents each decoder. It also provides a means to query the capabilities of each decoder. A given decoder may support several profile/mode pairs each with their own capabilities (such as maximum sample rate or bit rate) appropriate to that profile and mode pair. Therefore, this interface represents the capabilities of a particular decoder as a list of capability entries queriable by decoderID and capability entry index.

The set of audio decoders supported by the engine does not change during the lifetime of the engine though dynamic resource constraints may limit actual availability when an audio decoder is requested.

This interface is a mandated interface of engine objects (see section 7.2).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_AUDIODECODERCAPABILITIES;

struct XAAudioDecoderCapabilitiesItf_;
typedef const struct XAAudioDecoderCapabilitiesItf_
    * const * XAAudioDecoderCapabilitiesItf;

struct XAAudioDecoderCapabilitiesItf_ {
    XAresult (*GetAudioDecoders) (
        XAAudioDecoderCapabilitiesItf self,
        XAuint32 * pNumDecoders,
        XAuint32 * pDecoderIds
    );
    XAresult (*GetAudioDecoderCapabilities) (
        XAAudioDecoderCapabilitiesItf self,
        XAuint32 decoderId,
        XAuint32 * pIndex,
        XAAudioCodecDescriptor * pDescriptor
    );
};
```

Interface ID

deac0cc0-3995-11dc-8872-0002a5d5c51b

Defaults

Not applicable.

Methods

GetAudioDecoders			
XAresult (*GetAudioDecoders) (XAAudioDecoderCapabilitiesItf self, XAuint32 * pNumDecoders, XAuint32 * pDecoderIds);			
Description	Retrieves the available audio decoders.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumDecoders`	[in/out]	If `pDecoderIds` is `NULL`, `pNumDecoders` returns the number of decoders available. All implementations must have at least one decoder. If `pDecodersIds` is non-`NULL`, as an input `pNumDecoders` specifies the size of the `pDecoderIds` array and as an output it specifies the number of decoder IDs available within the `pDecoderIds` array.
	`pDecoderIds`	[out]	Array of audio decoders provided by the engine. Refer to `XA_AUDIOCODEC` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`GetAudioDecoderCapabilities()`		

GetAudioDecoderCapabilities			
XAresult (*GetAudioDecoderCapabilities) (**XAAudioDecoderCapabilitiesItf self,** **XAuint32 decoderId,** **XAuint32 * pIndex,** **XAAudioCodecDescriptor * pDescriptor** **);**			
Description	Queries for the audio decoder's capabilities.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`decoderId`	[in]	Identifies the supported audio decoder. Refer to `XA_AUDIOCODEC` macros.
	`pIndex`	[in/out]	If `pDescriptor` is `NULL`, `pIndex` returns the number of capabilities structures (one per profile/mode pair of the decoder). Each decoder must support at least one profile/mode pair and therefore have at least one Codec Descriptor. If `pDescriptor` is non-`NULL`, `pIndex` is an incrementing value used for enumerating capabilities. Supported index range is 0 to N-1, where N is the number of capabilities structures, one for each profile/mode pair of the decoder.
	pDescriptor	[out]	Pointer to structure defining the capabilities of the audio decoder. There is one structure per profile.mode pair of the decoder.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	This method outputs a structure that contains one or more pointers to arrays. The memory for these arrays shall be allocated by the OpenMAX AL implementation and shall not be deallocated by the application. The OpenMAX AL implementation shall keep the data contained within the arrays valid for the lifetime of this interface's host object. (The memory for the structure itself is allocated by the application and therefore shall be freed by the application.)		
See also	`GetAudioDecoders()`		

8.3 XAAudioEncoderItf

Description

This interface is used for setting the parameters to be used by an audio encoder. It is realized on a media object with audio encoding capabilities, such as a media recorder. Once the supported codecs have been enumerated using `XAAudioEncoderCapabilitiesItf` on the engine, the encoding settings can be set using this interface.

This interface is a mandated interface of Media Recorder objects (see section 7.5).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_AUDIOENCODER;

struct XAAudioEncoderItf_;
typedef const struct XAAudioEncoderItf_ * const * XAAudioEncoderItf;

struct XAAudioEncoderItf_ {
    XAresult (*SetEncoderSettings) (
        XAAudioEncoderItf self,
        XAAudioEncoderSettings * pSettings
    );
    XAresult (*GetEncoderSettings) (
        XAAudioEncoderItf self,
        XAAudioEncoderSettings * pSettings
    );
};
```

Interface ID

ebbab900-3997-11dc-891f-0002a5d5c51b

Defaults

No default settings are mandated.

Methods

SetEncoderSettings			
XAresult (*SetEncoderSettings) (XAAudioEncoderItf self, XAAudioEncoderSettings * pSettings);			
Description	Set audio encoder settings.		
Pre-conditions	RecordItf state shall be in stopped state.		
Parameters	`self`	[in]	Interface self-reference.
	`pSettings`	[in]	Specifies the audio encoder settings to be applied.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	None		
See also	`GetEncoderSettings()`		

GetEncoderSettings			
XAresult (*GetEncoderSettings) (XAAudioEncoderItf self, XAAudioEncoderSettings * pSettings);			
Description	Get audio encoder settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pSettings`	[out]	Specifies a pointer to the structure that will return the audio encoder settings.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`SetEncoderSettings()`		

8.4 XAAudioEncoderCapabilitiesItf

Description

This interface provides methods for querying the audio encoding capabilities of the media engine.

This interface provides a means of enumerating all audio encoders available on an engine where an encoderId represents each encoder. It also provides a means to query the capabilities of each encoder. A given encoder may support several profile/mode pairs, each with their own capabilities (such as maximum sample rate or bit rate) appropriate to that profile and mode pair. Therefore, this interface represents the capabilities of a particular encoder as a list of capability entries queriable by encoderID and capability entry index.

The set of audio encoders supported by the engine does not change during the lifetime of the engine though dynamic resource constraints may limit actual availability when an audio encoder is requested.

This interface is also mandated on the media recorder objects so that the effective capabilities of the encoder can be determined when it is part of the recorder object and source/sink combination. For example, an encoder might have an input sampling rate range of 8 kHz - 48 kHz overall, but when a recorder object is connected to a low-cost microphone, the effective sampling rate range of the encoder might only be 8 kHz - 16 kHz.

This interface is a mandated interface of engine objects (see section 7.2) and media recorder objects (see section 7.5).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_AUDIOENCODERCAPABILITIES;

struct XAAudioEncoderCapabilitiesItf_;
typedef const struct XAAudioEncoderCapabilitiesItf_
    * const * XAAudioEncoderCapabilitiesItf;

struct XAAudioEncoderCapabilitiesItf_ {
    XAresult (*GetAudioEncoders) (
        XAAudioEncoderCapabilitiesItf self,
        XAuint32 * pNumEncoders,
        XAuint32 * pEncoderIds
    );
    XAresult (*GetAudioEncoderCapabilities) (
        XAAudioEncoderCapabilitiesItf self,
        XAuint32 encoderId,
        XAuint32 * pIndex,
        XAAudioCodecDescriptor * pDescriptor
    );
};
```

Interface ID

e0f55ca0-f2d0-11df-b250-0002a5d5c51b

Defaults

Not applicable.

Methods

GetAudioEncoders			
XAresult (*GetAudioEncoders) (**XAAudioEncoderCapabilitiesItf self,** **XAuint32 * pNumEncoders,** **XAuint32 * pEncoderIds** **);**			
Description	Queries the supported audio encoders.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumEncoders`	[in/out]	If `pEncoderIds` is `NULL`, `pNumEncoders` returns the number of encoders available. Returns 0 if there are no encoders. If `pEncodersIds` is non-`NULL`, as an input `pNumEncoders` specifies the size of the `pEncoderIds` array and as an output it specifies the number of encoder IDs available within the `pEncoderIds` array.
	`pEncoderIds`	[out]	Array of audio encoders provided by the engine. Refer to `XA_AUDIOCODEC` macros

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	**PROFILE NOTES** *A Media Player/Recorder profile implementation must support at least one encoder.*		
See also	`GetAudioEncoder`Capabilities`()`		

GetAudioEncoderCapabilities			
XAresult (*GetAudioEncoderCapabilities) (**XAAudioEncoderCapabilitiesItf self,** **XAuint32 encoderId,** **XAuint32 * pIndex,** **XAAudioCodecDescriptor * pDescriptor** **);**			
Description	Queries for the audio encoder's capabilities.		
Pre-conditions	None		
Parameters	`Self`	[in]	Interface self-reference.
	`encoderId`	[in]	Identifies the supported audio encoder. Refer to `XA_AUDIOCODEC` macros.
	`pIndex`	[in/out]	If pDescriptor is `NULL`, `pIndex` returns the number of capabilities structures (one per profile/mode pair of the decoder). Each encoder must support at least one profile/mode pair and therefore have at least one Codec Descriptor. If pDescriptor is non-`NULL`, pIndex is an incrementing value used for enumerating capabilities structures. Supported index range is 0 to N-1, where N is the number of capabilities structures, one for each profile/mode pair of the encoder.
	pDescriptor	[out]	Pointer to structure defining the capabilities of the audio encoder. There is one structure per profile\mode of the encoder.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	This method outputs a structure that contains one or more pointers to arrays. The memory for these arrays shall be allocated by the OpenMAX AL implementation and shall not be deallocated by the application. The OpenMAX AL implementation shall keep the data contained within the arrays valid for the lifetime of this interface's host object. (The memory for the structure itself is allocated by the application and therefore shall be freed by the application.)		
See also	`GetAudioEncoders()`		

8.5 XAAudioIODeviceCapabilitiesItf

Description

This interface is for enumerating the audio I/O devices on the platform and for querying the capabilities and characteristics of each available audio I/O device.

This interface is supported on the engine object (see section 7.2). See section D.2 and section D.4 for examples using this interface.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_AUDIOIODEVICECAPABILITIES;

struct XAAudioIODeviceCapabilitiesItf_;
typedef const struct XAAudioIODeviceCapabilitiesItf_
    * const * XAAudioIODeviceCapabilitiesItf;

struct XAAudioIODeviceCapabilitiesItf_ {
    XAresult (*GetAvailableAudioInputs) (
        XAAudioIODeviceCapabilitiesItf self,
        XAint32 * pNumInputs,
        XAuint32 * pInputDeviceIDs
    );
    XAresult (*QueryAudioInputCapabilities) (
        XAAudioIODeviceCapabilitiesItf self,
        XAuint32 deviceID,
        XAAudioInputDescriptor * pDescriptor
    );
    XAresult (*RegisterAvailableAudioInputsChangedCallback) (
        XAAudioIODeviceCapabilitiesItf self,
        xaAvailableAudioInputsChangedCallback callback,
        void * pContext
    );
    XAresult (*GetAvailableAudioOutputs) (
        XAAudioIODeviceCapabilitiesItf self,
        XAint32 * pNumOutputs,
        XAuint32 * pOutputDeviceIDs
    );
    XAresult (*QueryAudioOutputCapabilities) (
        XAAudioIODeviceCapabilitiesItf self,
        XAuint32 deviceID,
        XAAudioOutputDescriptor * pDescriptor
    );
    XAresult (*RegisterAvailableAudioOutputsChangedCallback) (
        XAAudioIODeviceCapabilitiesItf self,
        xaAvailableAudioOutputsChangedCallback callback,
        void * pContext
    );
```

```
    XAresult (*RegisterDefaultDeviceIDMapChangedCallback) (
        XAAudioIODeviceCapabilitiesItf self,
        xaDefaultDeviceIDMapChangedCallback callback,
        void * pContext
    );
    XAresult (*GetAssociatedAudioInputs) (
        XAAudioIODeviceCapabilitiesItf self,
        XAuint32 deviceID,
        XAint32 * pNumAudioInputs,
        XAuint32 * pAudioInputDeviceIDs
    );
    XAresult (*GetAssociatedAudioOutputs) (
        XAAudioIODeviceCapabilitiesItf self,
        XAuint32 deviceID,
        XAint32 * pNumAudioOutputs,
        XAuint32 * pAudioOutputDeviceIDs
    );
    XAresult (*GetDefaultAudioDevices) (
        XAAudioIODeviceCapabilitiesItf self,
        XAuint32 defaultDeviceID,
        XAint32 * pNumAudioDevices,
        XAuint32 * pAudioDeviceIDs
    );
    XAresult (*QuerySampleFormatsSupported) (
        XAAudioIODeviceCapabilitiesItf self,
        XAuint32 deviceID,
        XAmilliHertz samplingRate,
        XAint32 * pSampleFormats,
        XAint32 * pNumOfSampleFormats
    );
};
```

Interface ID

2b276d00-f775-11db-a963-0002a5d5c51b

Defaults

I/O device capabilities vary widely from system to system. Defaults are not applicable.

Callbacks

xaAvailableAudioInputsChangedCallback			
typedef void (XAAPIENTRY * xaAvailableAudioInputsChangedCallback) (XAAudioIODeviceCapabilitiesItf caller, void * pContext, XAuint32 deviceID, XAint32 numInputs, XAboolean isNew);			
Description	This callback executes when the set of available audio input devices changes (as when a new Bluetooth headset is connected or a wired microphone is disconnected).		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`deviceID`	[in]	ID of the audio input device that has changed (that is, was either removed or added).
	`numInputs`	[in]	Updated number of available audio input devices.
	`isNew`	[in]	Set to `XA_BOOLEAN_TRUE` if the change was an addition of a newly available audio input device; `XA_BOOLEAN_FALSE` if an existing audio input device is no longer available.
Comments	The callback does not provide additional detail about the audio input device that has changed. In the case of an addition, it is up to the application to use `QueryAudioInputCapabilities()` to determine the full characteristics of the newly available audio input device.		
See Also	`QueryAudioInputCapabilities()`		

xaAvailableAudioOutputsChangedCallback

```
typedef void (XAAPIENTRY * xaAvailableAudioOutputsChangedCallback) (
    XAAudioIODeviceCapabilitiesItf caller,
    void * pContext,
    XAuint32 deviceID,
    XAint32 numOutputs,
    XAboolean isNew
);
```

Description	This callback executes when the set of available audio output devices changes (as when a new Bluetooth headset is connected or a wired headset is disconnected).		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`deviceID`	[in]	ID of the audio output device that has changed (that is, was either removed or added).
	`numOutputs`	[in]	Updated number of available audio output devices.
	`isNew`	[in]	Set to `XA_BOOLEAN_TRUE` if the change was an addition of a newly available audio output device; `XA_BOOLEAN_FALSE` if an existing audio output device is no longer available.
Comments	The callback does not provide additional details about the audio output device that has changed. In the case of an addition, it is up to the application to use `QueryAudioOutputCapabilities()` to determine the full characteristics of the newly-available audio output device.		
See Also	`QueryAudioOutputCapabilities`()		

xaDefaultDeviceIDMapChangedCallback			
typedef void (XAAPIENTRY * xaDefaultDeviceIDMapChangedCallback) (XAAudioIODeviceCapabilitiesItf caller, void * pContext, XAboolean isOutput, XAint32 numDevices);			
Description	This callback executes when the set of audio output devices mapped to XA_DEFAULTDEVICEID_AUDIOINPUT or XA_DEFAULTDEVICEID_AUDIOOUTPUT changes		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`isOutput`	[in]	If true, then devices mapped to XA_DEFAULTDEVICEID_AUDIOOUTPUT have changed, otherwise the devices mapped to XA_DEFAULTDEVICEID_AUDIOINPUT have changed.
	`numDevices`	[in]	New number of physical audio output devices to which `XA_DEFAULTDEVICEID_AUDIOOUTPUT or XA_DEFAULTDEVICEID_AUDIOINPUT` is now mapped (depending on the value of isOutput). Is always greater than or equal to 1.
Comments	The callback does not provide additional details about the audio output devices now mapped to the default device ID. It is up to the application to retrieve the device IDs and to use the device IDs to query the capabilities of each device. `numDevices` is included in the callback for the benefit of those applications who may not wish to send/receive their audio stream to/from more than one output device. Such applications can examine `numDevices` and opt to stop operation immediately if it is greater than 1, without needing to invoke other methods to get the new number of devices mapped to `XA_DEFAULTDEVICEID_AUDIOOUTPUT or XA_DEFAULTDEVICEID_AUDIOINPUT`.		
See Also	`QueryAudioOutputCapabilities`()		

Methods

GetAvailableAudioInputs			
XAresult (*GetAvailableAudioInputs) (XAAudioIODeviceCapabilitiesItf self, XAint32 * pNumInputs, XAuint32 * pInputDeviceIDs);			
Description	Gets the number and IDs of audio input devices currently available.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumInputs`	[in/out]	As an input, specifies the length of the `pInputDeviceIDs` array (ignored if `pInputDeviceIDs` is `NULL`). As an output, specifies the number of audio input device IDs available in the system. Returns 0 if no audio input devices are available in the system.
	`pInputDeviceIDs`	[out]	Array of audio input device IDs currently available in the system. This parameter is populated by the call with the array of input device IDs (provided that `pNumInputs` is equal to or greater than the number of actual input device IDs). If `pNumInputs` is less than the number of actual input device IDs, the error code `XA_RESULT_BUFFER_INSUFFICIENT` is returned.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_BUFFER_INSUFFICIENT`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Note that "available" implies those audio input devices that are active (that is, can accept input audio) and this number may be less than or equal to the total number of audio input devices in the system. For example, if a system has both an integrated microphone and a line-in jack, but the line-in jack is not connected to anything, the number of available audio inputs is only 1. Device IDs should not be expected to be contiguous. Device IDs are unique: the same device ID shall not be used for different device types.		
See Also	`GetAvailableAudioOutputs()`		

QueryAudioInputCapabilities			
XAresult (*QueryAudioInputCapabilities) (**XAAudioIODeviceCapabilitiesItf self,** **XAuint32 deviceID,** **XAAudioInputDescriptor * pDescriptor** **);**			
Description	Gets the capabilities of the specified audio input device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`deviceID`	[in]	ID of the audio input device.
	`pDescriptor`	[out]	Structure defining the capabilities of the audio input device.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_IO_ERROR`		

Comments	This method outputs a structure that contains one or more pointers to arrays. The memory for these arrays shall be allocated by the OpenMAX AL implementation and shall not be deallocated by the application. The OpenMAX AL implementation shall keep the data contained within the arrays valid for the lifetime of this interface's host object. (The memory for the structure itself is allocated by the application and therefore shall be freed by the application.)		
See Also	`QueryAudioOutputCapabilities(), QuerySampleFormatsSupported()`		

RegisterAvailableAudioInputsChangedCallback			
XAresult (*RegisterAvailableAudioInputsChangedCallback) (**XAAudioIODeviceCapabilitiesItf self,** **xaAvailableAudioInputsChangedCallback callback,** **void * pContext** **);**			
Description	Sets or clears `xaAvailableAudioInputsChangedCallback()`.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS.`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`xaAvailableAudioInputsChangedCallback()`		

GetAvailableAudioOutputs

```
XAresult (*GetAvailableAudioOutputs) (
    XAAudioIODeviceCapabilitiesItf self,
    XAint32 * pNumOutputs,
    XAuint32 * pOutputDeviceIDs
);
```

Description	Gets the number and IDs of audio output devices currently available.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumOutputs`	[in/out]	As an input, specifies the size of the `pOutputDeviceIDs` array (ignored if `pOutputDeviceIDs` is `NULL`). As an output, specifies the number of audio output devices currently available in the system. Returns 0 if no audio output devices are active in the system.
	`pOutputDeviceIDs`	[out]	Array of audio output device IDs that are currently available in the system. This parameter is populated by the call with the array of output device IDs (provided that `pNumOutputs` is equal to or greater than the number of actual device IDs).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_BUFFER_INSUFFICIENT`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Note that “available” implies those audio output devices that are active (that is, can render audio) and this number may be less than or equal to the total number of audio output devices on the system. For example, if a system has both an integrated loudspeaker and a 3.5mm headphone jack, but if the headphone jack is not connected to anything, the number of available audio outputs is only 1. Device IDs should not be expected to be contiguous. Device IDs are unique: the same device ID shall not be used for different device types.		
See Also	`GetAvailableAudioInputs()`		

QueryAudioOutputCapabilities

```
XAresult (*QueryAudioOutputCapabilities) (
    XAAudioIODeviceCapabilitiesItf self,
    XAuint32 deviceID,
    XAAudioOutputDescriptor * pDescriptor
);
```

Description	Gets the capabilities of the specified audio output device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`deviceID`	[in]	ID of the audio output device.
	`pDescriptor`	[out]	Structure defining the characteristics of the audio output device.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_IO_ERROR`		

Comments	This method outputs a structure that contains one or more pointers to arrays. The memory for these arrays shall be allocated by the OpenMAX AL implementation and shall not be deallocated by the application. The OpenMAX AL implementation shall keep the data contained within the arrays valid for the lifetime of this interface's host object. (The memory for the structure itself is allocated by the application and therefore shall be freed by the application.)		
See Also	`QueryAudioInputCapabilities(), QuerySampleFormatsSupported()`		

RegisterAvailableAudioOutputsChangedCallback

```
XAresult (*RegisterAvailableAudioOutputsChangedCallback) (
    XAAudioIODeviceCapabilitiesItf self,
    xaAvailableAudioOutputsChangedCallback callback,
    void * pContext
);
```

Description	Sets or clears `xaAvailableAudioOutputsChangedCallback()`.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`xaAvailableAudioOutputsChangedCallback()`		

RegisterDefaultDeviceIDMapChangedCallback			
XAresult (*RegisterDefaultDeviceIDMapChangedCallback) (**XAAudioIODeviceCapabilitiesItf self,** **xaDefaultDeviceIDMapChangedCallback callback,** **void * pContext** **);**			
Description	Sets or clears `xaDefaultDeviceIDMapChangedCallback()`.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`xaDefaultDeviceIDMapChangedCallback()`		

GetAssociatedAudioInputs

```
XAresult (*GetAssociatedAudioInputs) (
    XAAudioIODeviceCapabilitiesItf self,
    XAuint32 deviceID,
    XAint32 * pNumAudioInputs,
    XAuint32 * pAudioInputDeviceIDs
);
```

Description	This method returns an array of audio input devices physically associated with this audio I/O device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`deviceID`	[in]	ID of the input or output device .
	`pNumAudioInputs`	[in/out]	As an input, specifies the length of the `pAudioInputDeviceIDs` array (ignored if `pAudioInputDeviceIDs` is `NULL`). As an output, specifies the number of audio input device IDs associated with `deviceID`. Returns zero if there is no such association.
	`pAudioInputDeviceIDs`	[out]	Array of audio input device IDs. Should be ignored if `pNumAudioInputs` is zero – that is, if there are no associated audio inputs. This parameter is populated by the call with the array of input device IDs (provided that `pNumInputs` is equal to or greater than the number of actual input device IDs).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`
	`XA_RESULT_IO_ERROR`		

Comments	This method can be called on both audio input and audio output devices. It is useful for determining coupling of audio inputs and outputs on certain types of accessories. For example, it is helpful to know that microphone 01 is actually part of the same Bluetooth headset as speaker 03. Also, many car kits have multiple speakers and multiple microphones. Hence the need for an array of associated input devices. For applications that both accept and render audio, this method helps to determine whether an audio input and an audio output belong to the same physical accessory. An audio device cannot be associated with itself. So, in the example above, if this method were to be called with microphone 01 as the deviceID parameter, it would return an empty array, since there are no other inputs associated with microphone 01 on that Bluetooth headset. If this method is called with the special device IDs XA_DEFAULTDEVICEID_AUDIOINPUT and XA_DEFAULTDEVICEID_AUDIOOUTPUT, the result is undefined.		
See also	`GetDefaultAudioDevices()`		

GetAssociatedAudioOutputs

```
XAresult (*GetAssociatedAudioOutputs) (
    XAAudioIODeviceCapabilitiesItf self,
    XAuint32 deviceID,
    XAint32 * pNumAudioOutputs,
    XAuint32 * pAudioOutputDeviceIDs
);
```

Description	This method returns an array of audio output devices physically associated with this audio I/O device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`deviceID`	[in]	ID of the input or output device.
	`pNumAudioOutputs`	[in/out]	As an input, specifies the length of the `pAudioOutputDeviceIDs` array (ignored if `pAudioOutputDeviceIDs` is `NULL`). As an output, specifies the number of audio output device IDs associated with `deviceID`. Returns zero if there is no such association.
	`pAudioOutputDeviceIDs`	[out]	Array of audio output device IDs. Should be ignored if `pNumAudioOutputs` is zero (that is, there are no associated audio outputs). This parameter is populated by the call with the array of output device IDs (provided that `pNumAudioOutputs` is equal to or greater than the number of actual device IDs).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`
	`XA_RESULT_IO_ERROR`		

Comments	This method can be called on both audio input and audio output devices. It is useful for determining coupling of audio inputs and outputs on certain types of accessories. For example, it is helpful to know that microphone 01 is actually part of the same Bluetooth headset as speaker 03. Also, many car kits have multiple speakers and multiple microphones. Hence the need for an array of associated output devices. For applications that both accept and render audio, this method helps to determe whether an audio input and an audio output belong to the same physical accessory. An audio device cannot be associated with itself. So, in the example above, if this method were to be called with speaker 03 as the deviceID parameter, it would return an empty array, since there are no other outputs associated with speaker 03 on that Bluetooth headset. If this method is called with the special device IDs `XA_DEFAULTDEVICEID_AUDIOINPUT` and `XA_DEFAULTDEVICEID_AUDIOOUTPUT`, the result is undefined.		
See also	`GetDefaultAudioDevices()`		

GetDefaultAudioDevices

```
XAresult (*GetDefaultAudioDevices) (
   XAAudioIODeviceCapabilitiesItf self,
   XAuint32 defaultDeviceID,
   XAint32 * pNumAudioDevices,
   XAuint32 * pAudioDeviceIDs
);
```

Description	Gets the number of audio devices currently mapped to the given default device ID.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`defaultDeviceID`	[in]	ID of the default device (currently defined as `XA_DEFAULTDEVICEID_AUDIOOUTPUT` and `XA_DEFAULTDEVICEID_AUDIOINPUT` [see section 9.2.26]).
	`pNumAudioDevices`	[in/out]	As an input, specifies the length of the `pAudioDeviceIDs` array (ignored if `pAudioDeviceIDs` is `NULL`). As an output, specifies the number of audio device IDs mapped to the given `defaultDeviceID`.
	`pAudioDeviceIDs`	[out]	Array of audio device IDs that are currently mapped to the given `defaultDeviceID`. This parameter is populated by the call with the array of device IDs (provided that `pNumAudioDevices` is equal to or greater than the number of actual device IDs). If `pNumAudioDevices` is less than the number of actual mapped device IDs, the error code `XA_RESULT_BUFFER_INSUFFICIENT` is returned.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_BUFFER_INSUFFICIENT`
	`XA_RESULT_IO_ERROR`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The mapping of `defaultDeviceID` to the physical audio devices (represented by the device IDs) is implementation-dependent. The application can choose to be notified of the implementation-induced changes to this mapping by registering for the `xaDefaultDeviceIDMapChangedCallback()`.		
See Also	`RegisterDefaultDeviceIDMapChangedCallback()`, `GetAssociatedAudioInputs()`, `GetAssociatedAudioOutputs()`		

QuerySampleFormatsSupported			
XAresult (*QuerySampleFormatsSupported) (**XAAudioIODeviceCapabilitiesItf self,** **XAuint32 deviceID,** **XAmilliHertz samplingRate,** **XAint32 * pSampleFormats,** **XAint32 * pNumOfSampleFormats,** **);**			
Description	Gets an array of sample formats supported by the audio I/O device for the given sampling rate. The rationale here is that an audio I/O device might not support all sample formats at all sampling rates. Therefore, it is necessary to query the sample formats supported for each sampling rate of interest.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`deviceID`	[in]	ID of the audio I/O device
	`samplingRate`	[in]	Sampling rate for which the sampling formats are to be determined.
	`pSampleFormats`	[out]	Array of sample formats supported, as defined in the `XA_PCMSAMPLEFORMAT` macros. This parameter is populated by the call with the array of supported sample formats (provided that `pNumOfSampleFormats` is equal to or greater than the number of actual sample formats).
	`pNumOfSampleFormats`	[in/out]	As an input, specifies the length of the `pSampleFormats` array (ignored if `pSampleFormats` is `NULL`). As an output, specifies the number of sample formats supported.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`
	`XA_RESULT_IO_ERROR`		

Comments	None		
See Also	`QueryAudioInputCapabilities()`, `QueryAudioOutputCapabilities()`		

8.6 XACameraItf

Description

The camera interface is used for querying and changing the settings of a camera I/O device.

This interface is implicit on the camera I/O device object (see section 7.1).

See `XASnapshotItf` (see section 8.33) for a typical photographing call sequence example.

See section D.5 for an example using this interface.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_CAMERA;

struct XACameraItf_;
typedef const struct XACameraItf_ * const * XACameraItf;

struct XACameraItf_ {
    XAresult (*RegisterCallback) (
        XACameraItf self,
        xaCameraCallback callback,
        void * pContext
    );
    XAresult (*SetFlashMode) (
        XACameraItf self,
        XAuint32 flashMode
    );
    XAresult (*GetFlashMode) (
        XACameraItf self,
        XAuint32 * pFlashMode
    );
    XAresult (*IsFlashReady) (
        XACameraItf self,
        XAboolean * pReady
    );
    XAresult (*SetFocusMode) (
        XACameraItf self,
        XAuint32 focusMode,
        XAmillimeter manualSetting,
        XAboolean macroEnabled
    );
    XAresult (*GetFocusMode) (
        XACameraItf self,
        XAuint32 * pFocusMode,
        XAmillimeter * pManualSetting,
        XAboolean * pMacroEnabled
    );
```

```
XAresult (*SetFocusRegionPattern) (
    XACameraItf self,
    XAuint32 focusPattern,
    XAuint32 activePoints1,
    XAuint32 activePoints2
);
XAresult (*GetFocusRegionPattern) (
    XACameraItf self,
    XAuint32 * pFocusPattern,
    XAuint32 * pActivePoints1,
    XAuint32 * pActivePoints2
);
XAresult (*GetFocusRegionPositions) (
    XACameraItf self,
    XAuint32 * pNumPositionEntries,
    XAFocusPointPosition * pFocusPosition
);
XAresult (*GetFocusModeStatus) (
    XACameraItf self,
    XAuint32 * pFocusStatus,
    XAuint32 * pRegionStatus1,
    XAuint32 * pRegionStatus2
);
XAresult (*SetMeteringMode) (
    XACameraItf self,
    XAuint32 meteringMode
);
XAresult (*GetMeteringMode) (
    XACameraItf self,
    XAuint32 * pMeteringMode
);
XAresult (*SetExposureMode) (
    XACameraItf self,
    XAuint32 exposureMode,
    XAuint32 compensation
);
XAresult (*GetExposureMode) (
    XACameraItf self,
    XAuint32 * pExposureMode,
    XAuint32 * pCompensation
);
XAresult (*SetISOSensitivity) (
    XACameraItf self,
    XAuint32 isoSensitivity,
    XAuint32 manualSetting
);
XAresult (*GetISOSensitivity) (
    XACameraItf self,
    XAuint32 * pIsoSensitivity,
    XAuint32 * pManualSetting
);
XAresult (*SetAperture) (
    XACameraItf self,
    XAuint32 aperture,
    XAuint32 manualSetting
);
```

```
    XAresult (*GetAperture) (
        XACameraItf self,
        XAuint32 * pAperture,
        XAuint32 * pManualSetting
    );
    XAresult (*SetShutterSpeed) (
        XACameraItf self,
        XAuint32 shutterSpeed,
        XAmicrosecond manualSetting
    );
    XAresult (*GetShutterSpeed) (
        XACameraItf self,
        XAuint32 * pShutterSpeed,
        XAmicrosecond * pManualSetting
    );
    XAresult (*SetWhiteBalance) (
        XACameraItf self,
        XAuint32 whiteBalance,
        XAuint32 manualSetting
    );
    XAresult (*GetWhiteBalance) (
        XACameraItf self,
        XAuint32 * pWhiteBalance,
        XAuint32 * pManualSetting
    );
    XAresult (*SetAutoLocks) (
        XACameraItf self,
        XAuint32 locks
    );
    XAresult (*GetAutoLocks) (
        XACameraItf self,
        XAuint32 * pLocks
    );
    XAresult (*SetZoom) (
        XACameraItf self,
        XApermille zoom,
        XAboolean digitalEnabled,
        XAuint32 speed,
        XAboolean async
    );
    XAresult (*GetZoom) (
        XACameraItf self,
        XApermille * pZoom,
        XAboolean * pDigital
    );
};
```

Interface ID

0a614b80-d6d9-11df-b536-0002a5d5c51b

Defaults

No default settings are mandated.

Callbacks

xaCameraCallback			
typedef void (XAAPIENTRY * xaCameraCallback) (XACameraItf caller, void * pContext, XAuint32 eventId, XAuint32 eventData);			
Description	This method is used for camera event notifications.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`eventId`	[in]	Indicates the type of notification callback event being reported. Refer to `XA_CAMERACBEVENT` for a list of available events.
	`eventData`	[in]	Specifies additional information specific to a notification callback event. The contents of this parameter are dependent on the event being reported.
Comments	None		
See Also	`RegisterCallback()`		

Methods

RegisterCallback			
XAresult (*RegisterCallback) (**XACameraItf self,** **xaCameraCallback callback,** **void * pContext** **);**			
Description	Sets callback for camera event notifications.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Specifies the callback method.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Refer to `XA_CAMERACBEVENT` for the possible callback event notifications		
See Also	`xaCameraCallback()`		

SetFlashMode			
XAresult (*SetFlashMode) (**XACameraItf self,** **XAuint32 flashMode** **);**			
Description	Sets the camera flash setting.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`flashMode`	[in]	Specifies the camera flash setting. Refer to `XA_CAMERA_FLASHMODE`.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See Also	`GetFlashMode()`		

GetFlashMode			
XAresult (*GetFlashMode) (**XACameraItf self,** **XAuint32 * pFlashMode** **);**			
Description	Gets the camera flash setting.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pFlashMode`	[out]	Specifies the camera flash setting. Refer to `XA_CAMERA_FLASHMODE`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetFlashMode()`		

IsFlashReady			
XAresult (*IsFlashReady) (**XACameraItf self,** **XAboolean * pReady** **);**			
Description	Queries whether the flash is ready for use.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pReady`	[out]	Specifies whether the flash is ready.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`xaCameraCallback()`		

SetFocusMode			
XAresult (*SetFocusMode) (**XACameraItf self,** **XAuint32 focusMode,** **XAmillimeter manualSetting,** **XAboolean macroEnabled** **);**			
Description	Sets the camera focus mode.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`focusMode`	[in]	Specifies the camera focus mode. Refer to `XA_CAMERA_FOCUSMODE`.
	`manualSetting`	[in]	If the manual focus mode is enabled, this value specifies the manual setting. This parameter is ignored if manual focus mode is disabled.
	`macroEnabled`	[in]	Specifics whether macro mode is enabled.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See Also	`GetFocusMode(), XA_CAMERA_FOCUSMODE, GetSupportedFocusManualSettings()`		

GetFocusMode			
XAresult (*GetFocusMode) (**XACameraItf self,** **XAuint32 * pFocusMode,** **XAmillimeter * pManualSetting,** **XAboolean * pMacroEnabled** **);**			
Description	Gets the camera focus mode.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pFocusMode`	[out]	Specifies the camera focus mode. Refer to `XA_CAMERA_FOCUSMODE`.
	`pManualSetting`	[out]	If the manual focus mode is enabled, this value specifies the manual setting. This parameter is ignored if manual focus mode is disabled
	`pMacroEnabled`	[out]	Specifics whether the macro mode is enabled.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetFocusMode(), XA_CAMERA_FOCUSMODE, GetSupportedFocusManualSettings()`		

SetFocusRegionPattern			
XAresult (*SetFocusRegionPattern) (**XACameraItf self,** **XAuint32 focusPattern,** **XAuint32 activePoints1,** **XAuint32 activePoints2** **);**			
Description	Sets the camera focus region pattern.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`focusPattern`	[in]	Specifies the focus region pattern. Refer to `XA_CAMERA_FOCUSPOINTS`.
	`activePoints1`	[in]	Specifies the focus points to be used for the custom auto focus region pattern mode – `XA_CAMERA_FOCUSPOINTS_CUSTOM`. This parameter identifies the focus points ranging from 0 to 31. This parameter is ignored if manual focus mode is selected or non-custom focus points patterns are selected. This parameter is a bit-mapped representation of the focus points. Focus point 0 is identified by bit 0, focus point 1 is identified by bit 1 and so on. If a bit is set this indicates the point is to be used for autofocus. For example, to select only the center sixteen points (points 18 to 21, 26 to 29, 34 to 37 and 42 to 45): • `activePoints1` will have a value of 0x3C3C0000 • `activePoints2` will have a value of 0x00003C3C
	`activePoints2`	[in]	See description of `activePoints1`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`GetFocusRegionPattern(), XA_CAMERA_FOCUSPOINTS`		

GetFocusRegionPattern			
XAresult (*GetFocusRegionPattern) (**XACameraItf self,** **XAuint32 * pFocusPattern,** **XAuint32 * pActivePoints1,** **XAuint32 * pActivePoints2** **);**			
Description	Gets the camera focus region pattern.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pFocusPattern`	[out]	Specifies the focus region pattern. Refer to `XA_CAMERA_FOCUSPOINTS`.
	`pActivePoints1`	[out]	Specifies the focus points being used for the custom auto focus region pattern mode – `XA_CAMERA_FOCUSPOINTS_CUSTOM`. This parameter identifies the focus points, ranging from 0 to 31. This parameter is ignored if manual focus mode is selected or non-custom focus points patterns are selected. This parameter is a bit-mapped representation of the focus points. Focus point 0 is identified by bit 0, focus point 1 is identified by bit 1, and so on. If a bit is set this indicates the point is to be used for autofocus. For example, to select only the center sixteen points (points 18 to 21, 26 to 29, 34 to 37 and 42 to 45): • `pActivePoints1` will have a value of 0x3C3C0000 • `pActivePoints2` will have a value of 0x00003C3C
	`pActivePoints2`	[out]	See description of `activePoints1`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetFocusRegionPattern(), XA_CAMERA_FOCUSPOINTS`		

GetFocusRegionPositions			
XAresult (*GetFocusRegionPositions) (**XACameraItf self,** **XAuint32 * pNumPositionEntries,** **XAFocusPointPosition * pFocusPosition** **);**			
Description	Gets the camera focus region pattern's positioning and size for each point in the active focus pattern.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumPositionEntries`	[in/out]	As an input, this parameter specifies the size of the input buffer. As an output, this parameter specifies the number of position points being returned.
	`pFocusPosition`	[out]	Specifies the focus point's position information. The application provides this buffer and the buffer size needs to be a multiple of `XAFocusPointPosition`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetFocusRegionPattern(), XA_CAMERA_FOCUSPOINTS`		

GetFocusModeStatus

```
XAresult (*GetFocusModeStatus) (
    XACameraItf self,
    XAuint32 * pFocusStatus,
    XAuint32 * pRegionStatus1,
    XAuint32 * pRegionStatus2
);
```

Description	Gets the camera focus status.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pFocusStatus`	[out]	Specifies the camera focus mode status. Refer to `XA_CAMERA_FOCUSMODESTATUS`.
	`pRegionStatus1`	[out]	Specifies the individual focus region status. `pRegionStatus1` and `pRegionStatus2` are bit-mapped representation of the individual focus regions. `pRegionStatus1` identifies the focus points ranging from 0 to 31. `pRegionStatus2` identifies the focus points ranging from 32 to 63. If a bit is set, this indicates the region contains the focus status as described by `pFocusStatus`. Refer to `XA_CAMERA_FOCUSREGIONS` for a bit-mapped representation of each focus region.
	`pRegionStatus2`	[out]	See description of `pRegionStatus1`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetFocusMode(), XA_CAMERA_FOCUSMODESTATUS`		

SetMeteringMode			
XAresult (*SetMeteringMode) (**XACameraItf self,** **XAuint32 meteringMode** **);**			
Description	Sets the camera metering mode for exposure.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`meteringMode`	[in]	Specifies the camera metering mode. Refer to `XA_CAMERA_METERINGMODE`.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See Also	`GetMeteringMode(), XA_CAMERA_METERINGMODE`		

GetMeteringMode			
XAresult (*GetMeteringMode) (**XACameraItf self,** **XAuint32 * pMeteringMode** **);**			
Description	Gets the camera metering mode for exposure.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMeteringMode`	[out]	Specifies the camera metering mode. Refer to `XA_CAMERA_METERINGMODE`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetMeteringMode(), XA_CAMERA_METERINGMODE`		

SetExposureMode			
XAresult (*SetExposureMode) (**XACameraItf self,** **XAuint32 exposureMode,** **XAuint32 compensation** **);**			
Description	Sets the camera exposure mode.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`exposureMode`	[in]	Specifies the camera exposure mode. Refer to `XA_CAMERA_EXPOSUREMODE`.
	`compensation`	[in]	If the auto exposure mode setting is enabled, this value specifies the auto exposure compensation setting. This parameter is ignored if auto mode setting is not enabled. The parameter is in units of $1/10^{th}$ of EV compensation.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See Also	`GetExposure(), XA_CAMERA_EXPOSUREMODE`		

GetExposureMode			
XAresult (*GetExposureMode) (**XACameraItf self,** **XAuint32 * pExposureMode,** **XAuint32 * pCompensation** **);**			
Description	Gets the camera exposure mode.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pExposureMode`	[out]	Specifies the camera exposure mode. Refer to `XA_CAMERA_EXPOSUREMODE`.
	`pCompensation`	[out]	If the auto exposure mode setting is enabled, this value specifies the auto exposure compensation setting. The parameter is in units of $1/10^{th}$ of EV compensation.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetExposure(), XA_CAMERA_EXPOSUREMODE`		

SetISOSensitivity			
XAresult (*SetISOSensitivity) (**XACameraItf self,** **XAuint32 isoSensitivity,** **XAuint32 manualSetting** **);**			
Description	Sets the camera ISO sensitivity.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`sensitivity`	[in]	Specifies the camera ISO sensitivity mode. Refer to `XA_CAMERA_ISOSENSITIVITYMODE`.
	`manualSetting`	[in]	If the manual ISO sensitivity mode is enabled, this value specifies the manual setting. This parameter is ignored if manual ISO sensitivity mode is disabled. The parameter is an ISO value.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See Also	`GetISOSensitivity(), XA_CAMERA_ISOSENSITIVITYMODE, GetSupportedISOSensitivitySettings()`		

GetISOSensitivity			
XAresult (*GetISOSensitivity) (**XACameraItf self,** **XAuint32 * pIsoSensitivity,** **XAuint32 * pManualSetting** **);**			
Description	Gets the camera ISO sensitivity.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pSensitivity`	[out]	Specifies the camera ISO sensivitity mode. Refer to `XA_CAMERA_ISOSENSITIVITYMODE`.
	`pManualSetting`	[out]	If the manual ISO sensitivity mode is enabled, this value specifies the manual setting. If automatic ISO sensitivity mode is used, the exposure is locked and the device supports this, this value specifies the automatically determined ISO sensitivity; if exposure is not locked or exposing this value is not supported, this value is zero. The parameter is an ISO value.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetISOSensitivity(), XA_CAMERA_ISOSENSITIVITYMODE, GetSupportedISOSensitivitySettings()`		

SetAperture			
XAresult (*SetAperture) (**XACameraItf self,** **XAuint32 aperture,** **XAuint32 manualSetting** **);**			
Description	Sets the camera aperture.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`aperture`	[in]	Specifies the camera aperture mode. Refer to `XA_CAMERA_APERTUREMODE`.
	`manualSetting`	[in]	If the manual aperture mode is enabled, this value specifies the manual setting. This parameter is ignored if manual aperture mode is disabled. A setting of 100 is equal to an f-stop of 1.0.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See Also	`GetAperture()`, `XA_CAMERA_APERTUREMODE`, `GetSupportedApertureManualSettings()`		

GetAperture			
XAresult (*GetAperture) (**XACameraItf self,** **XAuint32 * pAperture,** **XAuint32 * pManualSetting** **);**			
Description	Gets the camera aperture.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pAperture`	[out]	Specifies the camera aperture mode. Refer to `XA_CAMERA_APERTUREMODE`.
	`pManualSetting`	[out]	If the manual aperture mode is enabled, this value specifies the manual setting. If automatic aperture mode is used, the exposure is locked and the device supports this, this value specifies the automatically determined aperture; if the exposure is not locked or exposing this value is not supported, this value is zero. A setting of 100 is equal to an f-stop of 1.0.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetAperture(), XA_CAMERA_APERTUREMODE, GetSupportedApertureManualSettings()`		

SetShutterSpeed			
XAresult (*SetShutterSpeed) (**XACameraItf self,** **XAuint32 shutterSpeed,** **XAmicrosecond manualSetting** **);**			
Description	Sets the camera shutter speed.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`shutterSpeed`	[in]	Specifies the camera shutter speed mode. Refer to `XA_CAMERA_SHUTTERSPEEDMODE`.
	`manualSetting`	[in]	If the manual shutter speed mode is enabled, this value specifies the manual setting. This parameter is ignored if manual shutter speed mode is disabled. The parameter is in units of microseconds.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See Also	`GetShutterSpeed(), XA_CAMERA_SHUTTERSPEEDMODE, GetSupportedWhiteBalanceManualSettings()`		

GetShutterSpeed			
XAresult (*GetShutterSpeed) (**XACameraItf self,** **XAuint32 * pShutterSpeed,** **XAmicrosecond * pManualSetting** **);**			
Description	Gets the camera shutter speed.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pShutterSpeed`	[out]	Specifies the camera shutter speed. Refer to `XA_CAMERA_SHUTTERSPEEDMODE`.
	`pManualSetting`	[out]	If the manual shutter speed mode is enabled, this value specifies the manual setting. If automatic shutter speed mode is used, the exposure is locked and the device supports this, this value specifies the automatically determined shutter speed; if the exposure is not locked or exposing this value is not supported, this value is zero. The parameter is in units of microseconds.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetShutterSpeed(), XA_CAMERA_SHUTTERSPEEDMODE, GetSupportedWhiteBalanceManualSettings()`		

SetWhiteBalance			
XAresult (*SetWhiteBalance) (XACameraItf self, XAuint32 whiteBalance, XAuint32 manualSetting);			
Description	Sets the camera white balance.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`whiteBalance`	[in]	Specifies the camera white balance mode. Refer to `XA_CAMERA_WHITEBALANCEMODE`.
	`manualSetting`	[in]	If the manual white balance mode is enabled, this value specifies the manual setting. This parameter is ignored if manual white balance mode is disabled. Parameter is in units of Kelvins.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See Also	`GetWhiteBalance(), XA_CAMERA_WHITEBALANCEMODE, GetSupportedFocusManualSettings()`		

GetWhiteBalance			
XAresult (*GetWhiteBalance) (XACameraItf self, XAuint32 * pWhiteBalance, XAuint32 * pManualSetting);			
Description	Gets the camera white balance.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pWhiteBalance`	[out]	Specifies the camera white balance mode. Refer to `XA_CAMERA_WHITEBALANCEMODE`.
	`pManualSetting`	[out]	If the manual white balance mode is enabled, this value specifies the manual setting. This parameter is ignored if manual white balance mode is disabled. Parameter is in units of Kelvins.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetWhiteBalance(), XA_CAMERA_WHITEBALANCEMODE, GetSupportedFocusManualSettings()`		

SetAutoLocks			
XAresult (*SetAutoLocks) (**XACameraItf self,** **XAuint32 locks** **);**			
Description	This method locks the given automatic camera settings. This method is typically called when the camera trigger is half-pressed. Locking is an asynchronous operation and results in related `xaCameraCallback()` calls with the events `XA_CAMERACBEVENT_FOCUSSTATUS`, `XA_CAMERACBEVENT_EXPOSURESTATUS` and/or `XA_CAMERACBEVENT_WHITEBALANCELOCKED` depending on which locks were requested.		
Pre-conditions	None		
Parameters	`self`	[in]	The camera interface.
	`locks`	[in]	A bitwise OR of the settings that will be locked. `XA_CAMERA_LOCK` macros define different locks. Zero can be used to unlock all the settings. The value must be one of the values given by `XACameraCapabilitiesItf::GetSupportedAutoLocks()`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	A lock doesn't have any effect if the corresponding setting is in manual mode.		
See also	`GetAutoLocks()`		

GetAutoLocks			
XAresult (*GetAutoLocks) (**XACameraItf self,** **XAuint32 * pLocks** **);**			
Description	This method gets the current state of the automatic camera setting locks.		
Pre-conditions	None		
Parameters	`self`	[in]	The camera interface.
	`pLocks`	[out]	A bitwise OR of the settings that are currently locked. See `XA_CAMERA_LOCK` macros for different locks.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`SetAutoLocks()`		

SetZoom			
XAresult (*SetZoom) (**XACameraItf self,** **XApermille zoom,** **XAboolean digitalEnabled,** **XAuint32 speed,** **XAboolean async** **);**			
Description	Sets the new zoom factor.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`zoom`	[in]	Specifies the zoom factor.
	`digitalEnabled`	[in]	If `XA_BOOLEAN_TRUE`, digital zoom and optical zoom is used; otherwise only optical zoom is used.
	`speed`	`[in]`	Hints the zooming speed. Accepted values are `XA_CAMERA_ZOOM_SLOW`, `XA_CAMERA_ZOOM_NORMAL`, `XA_CAMERA_ZOOM_FAST` and `XA_CAMERA_ZOOM_FASTEST`. This parameter is a hint and the exact actual zooming speed is implementation dependent. The exact speed might also be different when shooting video or still images.
	`async`	[in]	If `XA_BOOLEAN_FALSE`, the method will block until the requested zoom is completed. Otherwise, the method will return `XA_RESULT_SUCCESS`, and will be executed asynchronously. However, if the implementation is unable to initiate the asynchronous call `XA_RESULT_RESOURCE_ERROR` will be returned.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	When this method is executed asynchronously, the event `XA_CAMERACBEVENT_ZOOMSTATUS` is returned when zoom operation is completed.		
See Also	`GetZoom(),GetSupportedZoomSettings()`		

GetZoom			
XAresult (*GetZoom) (**XACameraItf self,** **XApermille * pZoom,** **XAboolean * pDigitalEnabled** **);**			
Description	Gets the current zoom factor.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pZoom`	[out]	Specifies the zoom factor.
	`pDigitalEnabled`	[out]	Specifies whether digital zoom is being used. `XA_BOOLEAN_TRUE` if digital zoom is used; `XA_BOOLEAN_FALSE` otherwise.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`SetZoom(),GetSupportedZoomSettings()`		

8.7 XACameraCapabilitiesItf

Description

This interface provides methods for querying the capabilities of camera I/O devices.

The set of cameras supported by the engine does not change during the lifetime of the engine, though dynamic resource constraints may limit actual availability when a camera is requested.

This interface is a mandated interface of engine objects (see section 7.2).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_CAMERACAPABILITIES;

struct XACameraCapabilitiesItf_;
typedef const struct XACameraCapabilitiesItf_
    * const * XACameraCapabilitiesItf;

struct XACameraCapabilitiesItf_ {
    XAresult (*GetCameraCapabilities) (
        XACameraCapabilitiesItf self,
        XAuint32 * pIndex,
        XAuint32 * pCameraDeviceID,
        XACameraDescriptor * pDescriptor
    );
    XAresult (*QueryFocusRegionPatterns) (
        XACameraCapabilitiesItf self,
        XAuint32 cameraDeviceID,
        XAuint32 * pPatternID,
        XAuint32 * pFocusPattern,
        XAuint32 * pCustomPoints1,
        XAuint32 * pCustomPoints2
    );
    XAresult (*GetSupportedAutoLocks) (
        XACameraCapabilitiesItf self,
        XAuint32 cameraDeviceID,
        XAuint32 * pNumCombinations,
        XAuint32 * pLocks
    );
    XAresult (*GetSupportedFocusManualSettings) (
        XACameraCapabilitiesItf self,
        XAuint32 cameraDeviceID,
        XAboolean macroEnabled,
        XAmillimeter * pMinValue,
        XAmillimeter * pMaxValue,
        XAuint32 * pNumSettings,
        XAmillimeter * pSettings
    );
    XAresult (*GetSupportedISOSensitivitySettings) (
        XACameraCapabilitiesItf self,
        XAuint32 cameraDeviceID,
        XAuint32 * pMinValue,
        XAuint32 * pMaxValue,
        XAuint32 * pNumSettings,
        XAuint32 * pSettings
    );
    XAresult (*GetSupportedApertureManualSettings) (
        XACameraCapabilitiesItf self,
        XAuint32 cameraDeviceID,
        XAuint32 * pMinValue,
        XAuint32 * pMaxValue,
        XAuint32 * pNumSettings,
        XAuint32 * pSettings
    );
```

```
    XAresult (*GetSupportedShutterSpeedManualSettings) (
        XACameraCapabilitiesItf self,
        XAuint32 cameraDeviceID,
        XAmicrosecond * pMinValue,
        XAmicrosecond * pMaxValue,
        XAuint32 * pNumSettings,
        XAmicrosecond * pSettings
    );
    XAresult (*GetSupportedWhiteBalanceManualSettings) (
        XACameraCapabilitiesItf self,
        XAuint32 cameraDeviceID,
        XAuint32 * pMinValue,
        XAuint32 * pMaxValue,
        XAuint32 * pNumSettings,
        XAuint32 * pSettings
    );
    XAresult (*GetSupportedZoomSettings) (
        XACameraCapabilitiesItf self,
        XAuint32 cameraDeviceID,
        XAboolean digitalEnabled,
        XAboolean macroEnabled,
        XApermille * pMaxValue,
        XAuint32 * pNumSettings,
        XApermille * pSettings,
        XAboolean * pSpeedSupported
    );
};
```

Interface ID

3af32780-d6db-11df-bda5-0002a5d5c51b

Defaults

Not applicable.

Methods

GetCameraCapabilities

```
XAresult (*GetCameraCapabilities) (
    XACameraCapabilitiesItf self,
    XAuint32 * pIndex,
    XAuint32 * pCameraDeviceID,
    XACameraDescriptor * pDescriptor
);
```

Description	Queries the camera device for its capabilities.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pIndex`	[in/out]	As an input, specifies which camera device to obtain the capabilities of, the supported range is [0, *n-1*), where *n* is the number of camera devices available. If pDescriptor is NULL, then as an output, specifies the number of camera devices available in the system. Returns 0 if no camera devices are available.
	`pCameraDeviceId`	[in/out]	If `pIndex` is non-NULL then returns the camera device ID corresponding to camera device `pIndex`. If pIndex is NULL then, as an input, specifies which camera device to obtain the capabilities of *(*`XA_DEFAULTDEVICEID_CAMERA` can be used to determine the default camera device's capabilities).
	`pDescriptor`	[out]	Structure defining the capabilities of the camera.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	An application can determine the number of camera devices by calling this method with `pDescriptor` set to NULL and examining pIndex. The application can then determine the capabilties of all the camera devices by calling this method multiple times with `pIndex` pointing to each different indexes from 0 up to one less than the number of camera devices. A camera is selected using the `CreateCameraDevice()` method. This method outputs a structure that contains one or more pointers to arrays. The memory for these arrays shall be allocated by the OpenMAX AL implementation and shall not be deallocated by the application. The OpenMAX AL implementation shall keep the data contained within the arrays valid for the lifetime of this interface's host object. (The memory for the structure itself is allocated by the application and therefore shall be freed by the application.)		
See also	XA_DEFAULTDEVICEID_CAMERA [see section 9.2.26]		

QueryFocusRegionPatterns

```
XAresult (*QueryFocusRegionPatterns) (
    XACameraCapabilitiesItf self,
    XAuint32 cameraDeviceID,
    XAuint32 * pPatternID,
    XAuint32 * pFocusPattern,
    XAuint32 * pCustomPoints1,
    XAuint32 * pCustomPoints2
);
```

Description	Queries the camera device for its focus region patterns.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`cameraDeviceID`	[in]	Camera device ID.
	`pPatternID`	[in/out]	If `pFocusPattern` is `NULL`, `pPatternID` returns the number of focus region patterns supported by the camera. Returns 0 if no focus region patterns are supported. If `pFocusPattern` is non-`NULL`, `pPatternID` is an incrementing value used for enumerating focus region patterns. Supported index range is 0 to N-1, where N is the number of focus region patterns.
	`pFocusPattern`	[out]	Focus point pattern used by the camera. See `XA_FOCUSPOINTS` macros.
	`pCustomPoints1`	[out]	Identifies the focus points available for the custom focus region pattern – `XA_FOCUSPOINTS_CUSTOM`. The parameter returns points that are selectable, and the points are identified as a bit array. This parameter is to be ignored if not used to query for the custom focus points information.
	`pCustomPoints2`	[out]	See description of `pCustomPoints1`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

GetSupportedAutoLocks

```
XAresult (*GetSupportedAutoLocks) (
    XACameraCapabilitiesItf self,
    XAuint32 cameraDeviceID,
    XAuint32 * pNumCombinations,
    XAuint32 * pLocks
);
```

Description	This method gets the supported combinations of automatic camera setting locks.

GetSupportedAutoLocks			
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`cameraDeviceID`	[in]	Camera Device ID.
	`pNumCombinations`	[in/out]	If `pLocks` is `NULL`, `pNumCombinations` returns the number of supported lock combinations. If `pLocks` is non-`NULL`, `pNumCombinations` is length of the pLocks array.
	`pLocks`	[out]	Returns an array of supported lock state combinations (bitwise ORs of `XA_CAMERA_LOCK` macros). `pLocks` may be NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If no lock state combinations are supported, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`. Not all the cameras support locking all these parameters pre-exposure. Furthermore, some combinations of otherwise supported locks might not be supported; for example a camera might support only locking focus and exposure together at once, but not separately, and no other locks. In that example case this method would return just three (`XA_CAMERA_LOCK_AUTOFOCUS` | `XA_CAMERA_LOCK_AUTOEXPOSURE`) and zero. The memory for the pLocks array shall be allocated by the OpenMAX AL application.		
See also	None		

GetSupportedFocusManualSettings

```
XAresult (*GetSupportedFocusManualSettings) (
    XACameraCapabilitiesItf self,
    XAuint32 cameraDeviceID,
    XAboolean macroEnabled,
    XAmillimeter * pMinValue,
    XAmillimeter * pMaxValue,
    XAuint32 * pNumSettings,
    XAmillimeter * pSettings
);
```

Description	This method gets the supported manual focus settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`cameraDeviceID`	[in]	Camera Device ID.
	`macroEnabled`	[in]	If `XA_BOOLEAN_TRUE`, returns focus settings for macro mode. If `XA_BOOLEAN_FALSE`, returns focus settings for normal mode.
	`pMinValue`	[out]	Identifies the minimum manual focus setting supported.
	`pMaxValue`	[out]	Identifies the maximum manual focus setting supported. A value of `0xFFFFFFFF` indicates infinity.
	`pNumSettings`	[in/out]	If `pSettings` is `NULL`, `pNumSettings` returns the number of supported manual focus settings. If the available manual settings are continuous from `pMinValue` to `pMaxValue`, `pNumSettings` returns `0`. If `pSettings` is non-`NULL` and a non-continuous range is supported, `pNumSettings` is length of the `pSettings` array.
	`pSettings`	[out]	Returns an array of supported focus settings. `pSettings` may be `NULL`. The array of values returned must include `pMinValue` and `pMaxValue`. A value of `0xFFFFFFFF` indicates infinity.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The value of `0xFFFFFFFF` for infinity should not be used with a continuous range of focus settings. If manual focus settings are unsupported, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`. The memory for the pSettings array shall be allocated by the OpenMAX AL application.		
See also	None		

GetSupportedISOSensitivitySettings			
XAresult (*GetSupportedISOSensitivitySettings) (**XACameraCapabilitiesItf self,** **XAuint32 cameraDeviceID,** **XAuint32 * pMinValue,** **XAuint32 * pMaxValue,** **XAuint32 * pNumSettings,** **XAuint32 * pSettings** **);**			
Description	This method gets the supported manual ISO settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`cameraDeviceID`	[in]	Camera Device ID.
	`pMinValue`	[out]	Identifies the minimum manual ISO setting supported in units of ISO values.
	`pMaxValue`	[out]	Identifies the maximum manual ISO supported in units of ISO values.
	`pNumSettings`	[in/out]	If `pSettings` is `NULL`, `pNumSettings` returns the number of supported manual ISO sensitivity settings. If the available manual settings are continuous from `pMinValue` to `pMaxValue`, `pNumSettings` returns `0`. If `pSettings` is non-`NULL` and a non-continuous range is supported, `pNumSettings` is length of the `pSettings` array.
	`pSettings`	[out]	Returns an array of supported ISO sensitivity settings. `pSettings` may be `NULL`. The values identified in the array are in units of ISO values. The array of values returned must include `pMinValue` and `pMaxValue`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If manual ISO settings are unsupported, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`. The memory for the pSettings array shall be allocated by the OpenMAX AL application.		
See also	None		

GetSupportedApertureManualSettings

```
XAresult (*GetSupportedApertureManualSettings) (
    XACameraCapabilitiesItf self,
    XAuint32 cameraDeviceID,
    XAuint32 * pMinValue,
    XAuint32 * pMaxValue,
    XAuint32 * pNumSettings,
    XAuint32 * pSettings
);
```

Description	This method gets the supported manual aperture settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`cameraDeviceID`	[in]	Camera Device ID.
	`pMinValue`	[out]	Identifies the minimum manual aperture setting supported. A setting of 100 is equal to an f-stop of 1.0.
	`pMaxValue`	[out]	Identifies the maximum manual aperture setting supported. A setting of 100 is equal to an f-stop of 1.0.
	`pNumSettings`	[in/out]	If `pSettings` is `NULL`, `pNumSettings` returns the number of supported manual aperture settings. If the available manual settings are continuous from `pMinValue` to `pMaxValue`, `pNumSettings` returns `0`. If `pSettings` is non-`NULL` and a non-continuous range is supported, `pNumSettings` is length of the `pSettings` array.
	`pSettings`	[out]	Returns an array of supported aperture settings. `pSettings` may be `NULL`. The array of values returned must include `pMinValue` and `pMaxValue`. A setting of 100 is equal to an f-stop of 1.0.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If manual aperture settings are unsupported, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`. The memory for the pSettings array shall be allocated by the OpenMAX AL application.		
See also	None		

GetSupportedShutterSpeedManualSettings

```
XAresult (*GetSupportedShutterSpeedManualSettings) (
    XACameraCapabilitiesItf self,
    XAuint32 cameraDeviceID,
    XAmicrosecond * pMinValue,
    XAmicrosecond * pMaxValue,
    XAuint32 * pNumSettings,
    XAmicrosecond * pSettings
);
```

Description	This method gets the supported manual shutter speed settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`cameraDeviceID`	[in]	Camera Device ID.
	`pMinValue`	[out]	Identifies the minimum manual shutter speed setting supported.
	`pMaxValue`	[out]	Identifies the maximum manual shutter speed setting supported.
	`pNumSettings`	[in/out]	If `pSettings` is `NULL`, `pNumSettings` returns the number of supported manual shutter speed settings. If the available manual settings are continuous from `pMinValue` to `pMaxValue`, `pNumSettings` returns `0`. If `pSettings` is non-`NULL` and a non-continuous range is supported, `pNumSettings` is length of the `pSettings` array.
	`pSettings`	[out]	Returns an array of supported shutter speed settings. `pSettings` may be `NULL`. The array of values returned must include `pMinValue` and `pMaxValue`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If manual shutter speed settings are unsupported, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`. The memory for the pSettings array shall be allocated by the OpenMAX AL application.		
See also	None		

GetSupportedWhiteBalanceManualSettings

```
XAresult (*GetSupportedWhiteBalanceManualSettings) (
    XACameraCapabilitiesItf self,
    XAuint32 cameraDeviceID,
    XAuint32 * pMinValue,
    XAuint32 * pMaxValue,
    XAuint32 * pNumSettings,
    XAuint32 * pSettings
);
```

Description	This method gets the supported manual white balance settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`cameraDeviceID`	[in]	Camera Device ID.
	`pMinValue`	[out]	Identifies the minimum manual white balance setting supported in units of Kelvins.
	`pMaxValue`	[out]	Identifies the maximum manual white balance setting supported in units of Kelvins.
	`pNumSettings`	[in/out]	If `pSettings` is `NULL`, `pNumSettings` returns the number of supported manual white balance settings. If the available manual settings are continuous from `pMinValue` to `pMaxValue`, `pNumSettings` returns `0`. If `pSettings` is non-`NULL` and a non-continuous range is supported, `pNumSettings` is length of the `pSettings` array.
	`pSettings`	[out]	Returns an array of supported white balance settings. `pSettings` may be `NULL`. The values identified in the array are in units of Kelvins. The array of values returned must include `pMinValue` and `pMaxValue`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If manual white balance settings are unsupported, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`. The memory for the pSettings array shall be allocated by the OpenMAX AL application.		
See also	None		

GetSupportedZoomSettings

```
XAresult (*GetSupportedZoomSettings) (
    XACameraCapabilitiesItf self,
    XAuint32 cameraDeviceID,
    XAboolean digitalEnabled,
    XAboolean macroEnabled,
    XApermille * pMaxValue,
    XAuint32 * pNumSettings,
    XApermille * pSettings,
    XAboolean * pSpeedSupported
);
```

Description	This method gets the supported zoom settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`cameraDeviceID`	[in]	Camera Device ID.
	`digitalEnabled`	[in]	If `XA_BOOLEAN_TRUE`, returns zoom settings when optical and digital zoom is enabled. If `XA_BOOLEAN_FALSE`, returns zoom settings when only optical zoom is enabled.
	`macroEnabled`	[in]	If `XA_BOOLEAN_TRUE`, returns zoom settings for macro mode. If `XA_BOOLEAN_FALSE`, returns zoom settings for normal mode.
	`pMaxValue`	[out]	Identifies the maximum zoom setting supported.
	`pNumSettings`	[in/out]	If `pSettings` is `NULL`, `pNumSettings` returns the number of supported zoom settings. If the available settings are continuous from `1000` to `pMaxValue`, `pNumSettings` returns `0`. If `pSettings` is non-`NULL` and a non-continuous range is supported, `pNumSettings` is length of the `pSettings` array.
	`pSettings`	`[out]`	Returns an array of supported zoom settings. `pSettings` may be `NULL`. The array of values returned must include `1000` and `pMaxValue`.
	`pSpeedSupported`	`[out]`	Returns `XA_BOOLEAN_TRUE` if zoom speed parameter in `XACameraItf::SetZoom` is supports; returns `XA_BOOLEAN_FALSE` otherwise.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The minimum zoom settings is 1000‰. If zoom is not supported, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`. The memory for the pSettings array shall be allocated by the OpenMAX AL application.		

GetSupportedZoomSettings	
See also	None

8.8 XAConfigExtensionsItf

Description

This interface provides a mechanism for an application to set and query both the codec and non-codec configurations of the underlying media engine (such as audio/video/image). These configuration parameters are in the form of key-value pairs. As such, the method signatures do not assume any vendor-specific or platform-specific knowledge of the underlying media engine or codecs. The methods of this interface have been designed such that they can be used to get/set the parameters for any OpenMAX AL object in a vendor-specific manner. Therefore, the usage of this interface is not limited to media engines or codecs. It applicable to all OpenMAX AL objects.

This interface can be exposed on any OpenMAX AL object.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_CONFIGEXTENSION;

struct XAConfigExtensionsItf_;
typedef const struct XAConfigExtensionsItf_
    * const * XAConfigExtensionsItf;

struct XAConfigExtensionsItf_ {
    XAresult (*SetConfiguration) (
        XAConfigExtensionsItf self,
        const XAchar * pConfigKey,
        XAuint32 valueSize,
        const void * pConfigValue
    );
    XAresult (*GetConfiguration) (
        XAConfigExtensionsItf self,
        const XAchar * pConfigKey,
        XAuint32 * pValueSize,
        void * pConfigValue
    );
};
```

Interface ID

2879ed80-d6dd-11df-87db-0002a5d5c51b

Defaults

None

Methods

SetConfiguration			
XAresult (*SetConfiguration) (**XAConfigExtensionsItf self,** **const XAchar * pConfigKey,** **XAuint32 valueSize;** **const void * pConfigValue** **);**			
Description	Sets the configuration as a key-value pair		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pConfigKey`	[in]	String representing the "key" – the parameter/attribute name of the configuration.
	`valueSize`	[in]	The size of the value referenced by `pConfigValue`, in bytes.
	`pConfigValue`	[in]	Address of the parameter/attribute being set.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The configValue input parameter is passed by reference. For example, this method could be used to set the RTSP proxy IP address and port number (for example, 123.213.123.5:80), or the bearer-specific bandwidth limits (for example, 900-1800 MHz). It is up to the underlying object to appropriately parse the key-value pair and make sense of the parameter setting.		

GetConfiguration

```
XAresult (*GetConfiguration) (
   XAConfigExtensionItf self,
   const XAchar * pConfigKey,
   XAuint32 * pValueSize,
   void * pConfigValue
);
```

Description	Gets the configuration setting as a key-value pair		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pConfigKey`	[in]	String representing the "key" – the name of the parameter/attribute being queried. If `configKey` is not recognized as a valid parameter/attributes of the underlying object `XA_RESULT_PARAMETER_INVALID` is return.
	`pValueSize`	[in/out]	Address of the size of the memory block passed as `pConfigValue`.
	`pConfigValue`	[out]	Address of the value of the parameter/attribute that is returned. If the size of the memory block passed as `pConfigValue` is too small to return the entire value, `XA_RESULT_PARAMETER_INVALID` is returned.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If the memory area specified by `pConfigValue` and `pValueSize` is too small to receive the entire value, only the first `pValueSize` bytes will be returned in `pConfigValue`. `pValueSize` will be set to the minimum size required for the call to succeed. The `pConfigValue` output parameter is passed by reference. For example, this method could be used for querying the RTSP proxy IP address and port number (123.213.123.5:80), or the bearer-specific bandwidth limits (900-1800 MHz). It is up to the underlying object to appropriately parse the key string and return the corresponding parameter setting, in the appropriate format. An error is returned if the key is not recognized by the underlying object.		

8.9 XADeviceVolumeItf

Description

This interface exposes controls for manipulating the volume of specific audio input and audio output devices. The units used for setting and getting the volume can be in millibels or as arbitrary volume steps; the units supported by the device can be queried with GetVolumeScale method.

Support for this interface is optional but where supported, this interface should be exposed on the engine object.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_DEVICEVOLUME;

struct XADeviceVolumeItf_;
typedef const struct XADeviceVolumeItf_ * const * XADeviceVolumeItf;

struct XADeviceVolumeItf_ {
    XAresult (*GetVolumeScale) (
        XADeviceVolumeItf self,
        XAuint32 deviceID,
        XAint32 * pMinValue,
        XAint32 * pMaxValue,
        XAboolean * pIsMillibelScale
    );
    XAresult (*SetVolume) (
        XADeviceVolumeItf self,
        XAuint32 deviceID,
        XAint32 volume
    );
    XAresult (*GetVolume) (
        XADeviceVolumeItf self,
        XAuint32 deviceID,
        XAint32 * pVolume
    );
};
```

Interface ID

4bb44020-f775-11db-ad03-0002a5d5c51b

Defaults

The default volume setting of each device should be audible.

Methods

GetVolumeScale			
XAresult (*GetVolumeScale) (**XADeviceVolumeItf self,** **XAuint32 deviceID,** **XAint32 * pMinValue,** **XAint32 * pMaxValue,** **XAboolean * pIsMillibelScale** **);**			
Description	Gets the properties of the volume scale supported by the given device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`deviceID`	[in]	Audio input or output device's identifier.
	`pMinValue`	[out]	The smallest supported volume value of the device.
	`pMaxValue`	[out]	The greatest supported volume value of the device.
	`pIsMillibelScale`	[out]	If true, the volume values used by GetVolume, SetVolume and this method are in millibel units; if false, the volume values are in arbitrary volume steps.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTROL_LOST`		

Comments	This method may return `XA_RESULT_FEATURE_UNSUPPORTED` if the specified device does not support changes to its volume. The scale is always continuous and device-specific. It could be, for example, [0, 15] if arbitrary volume steps are used or [-32768, 0] if millibels are used.		
See also	`XAAudioIODeviceCapabilitiesItf()`, `XAOutputMixItf()`		

SetVolume			
XAresult (*SetVolume) (**XADeviceVolumeItf self,** **XAuint32 deviceID,** **XAint32 volume** **);**			
Description	Sets the device's volume.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`deviceID`	[in]	Device identifier.
	`volume`	[in]	The new volume setting. The valid range is continuous and its boundaries can be queried from GetVolumeScale method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTROL_LOST`		

Comments	The minimum and maximum supported volumes are device-dependent. This method may fail if the specified device does not support changes to its volume or the volume is outside the range supported by the device.		
See also	`XAAudioIODeviceCapabilitiesItf`, `XAOutputMixItf`		

GetVolume			
XAresult (*GetVolume) (**XADeviceVolumeItf self,** **XAuint32 deviceID,** **XAint32 * pVolume** **);**			
Description	Gets the device's volume.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`deviceID`	[in]	Device identifier.
	`pVolume`	[out]	Pointer to a location to receive the object's volume setting. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	XA_RESULT_FEATURE_UNSUPPORTED is returned if the specified device does not support changes to its volume.		

8.10 XADynamicInterfaceManagementItf

Description

The `XADynamicInterfaceManagementItf` interface provides methods for handling interface exposure on an object after the creation and realization of the object. The primary method for exposing interfaces on an object is by listing them in the engine object's creation methods (see section 8.13).

`XADynamicInterfaceManagementItf` is an implicit interface of all object types. Please refer to section 3.1.7 for details about how dynamically exposed interfaces work with the object states and other exposed interfaces.

This interface is supported on all objects (see section 7).

Defaults

No dynamic interfaces are exposed.

No callback is registered.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_DYNAMICINTERFACEMANAGEMENT;

struct XADynamicInterfaceManagementItf_;
typedef const struct XADynamicInterfaceManagementItf_
    * const * XADynamicInterfaceManagementItf;

struct XADynamicInterfaceManagementItf_ {
    XAresult (*AddInterface) (
        XADynamicInterfaceManagementItf self,
        const XAInterfaceID iid,
        XAboolean async
    );
    XAresult (*RemoveInterface) (
        XADynamicInterfaceManagementItf self,
        const XAInterfaceID iid
    );
    XAresult (*ResumeInterface) (
        XADynamicInterfaceManagementItf self,
        const XAInterfaceID iid,
        XAboolean async
    );
```

```
    XAresult (*RegisterCallback) (
        XADynamicInterfaceManagementItf self,
        xaDynamicInterfaceManagementCallback callback,
        void * pContext
    );
};
```

Interface ID

6e2340c0-f775-11db-85da-0002a5d5c51b

Callbacks

xaDynamicInterfaceManagementCallback			
`typedef void (XAAPIENTRY * xaDynamicInterfaceManagementCallback) ( XADynamicInterfaceManagementItf caller, void * pContext, XAuint32 event, XAresult result, const XAInterfaceID iid );`			
Description	A callback function, notifying of a runtime error, termination of an asynchronous call or change in a dynamic interface's resources.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`event`	[in]	One of the dynamic interface management event macros. See `XA_DYNAMIC_ITF_EVENT` macros.

	`result`	[in]	Contains either the error code, if `event` is `XA_DYNAMIC_ITF_EVENT_RUNTIME_ERROR`, or the asynchronous function return code, if `event` is `XA_DYNAMIC_ITF_EVENT_ASYNC_TERMINATION.` The result may be:
			`XA_RESULT_SUCCESS`
			`XA_RESULT_PARAMETER_INVALID`
			`XA_RESULT_MEMORY_FAILURE`

	`iid`	[in]	Interface type ID that the event affects.
Comments	Please note the restrictions applying to operations performed from within callback context in section 3.3.		
See also	`RegisterCallback()`		

Methods

AddInterface			
XAresult (*AddInterface) (**XADynamicInterfaceManagementItf self,** **const XAInterfaceID iid,** **XAboolean async** **);**			
Description	Optionally asynchronous method for exposing an interface on an object. In asynchronous mode the success or failure of exposing the interface will be sent to the registered callback function.		
Pre-conditions	Interface has not been exposed.		
Parameters	`self`	[in]	Interface self-reference.
	`iid`	[in]	Valid interface type ID.
	`async`	[in]	If `XA_BOOLEAN_FALSE`, the method will block until termination. Otherwise, the method will return `XA_RESULT_SUCCESS`, and will be executed asynchronously. However, if the implementation is unable to initiate the asynchronous call `XA_RESULT_RESOURCE_ERROR` will be returned.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	When successful, the interface is exposed on the object and the interface pointer can be obtained by `XAObjectItf::GetInterface()`. Adding the interface to the object acquires the resources required for its functionality. The operation may fail if insufficient resources are available. In such a case, the application may wait until resources become available (`XA_DYNAMIC_ITF_EVENT_RESOURCE_AVAILABLE`), and then resume the interface. Additionally, the application may increase the object's priority, thus increasing the likelihood that the object will steal another object's resources. Adding an interface that is already exposed will result in a return value of `XA_RESULT_PRECONDITIONS_VIOLATED`.		
See also	`XAObjectItf::GetInterface()`		

RemoveInterface			
XAresult (*RemoveInterface) (**XADynamicInterfaceManagementItf self,** **const XAInterfaceID iid** **);**			
Description	Synchronous method for removing a dynamically exposed interface on the object. This method is supported in all object states.		
Pre-conditions	Interface has been exposed.		
Parameters	`self`	[in]	Interface self-reference.
	`iid`	[in]	Valid interface type ID that has been exposed on this object by use of the `AddInterface()` method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	An object that is in Suspended or Unrealized states waits also for resources for dynamically managed interfaces before sending a resources available event. By removing a dynamic interface in Unrealized or Suspended state, the object does not wait for resources for that dynamic interface. Removing an interface that is not exposed will result in a return value of `XA_RESULT_PRECONDITIONS_VIOLATED`.		
See also	None		

ResumeInterface

```
XAresult (*ResumeInterface) (
    XADynamicInterfaceManagementItf self,
    const XAInterfaceID iid,
    XAboolean async
);
```

Description	Optionally asynchronous method for resuming a dynamically exposed interface on the object.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`iid`	[in]	Valid interface type ID that has been exposed on this object by use of the `AddInterface()` method.
	`aync`	[in]	If `XA_BOOLEAN_FALSE`, the method will block until termination. Otherwise, the method will return `XA_RESULT_SUCCESS`, and will be executed asynchronously. However, if the implementation is unable to initiate the asynchronous call `XA_RESULT_RESOURCE_ERROR` will be returned.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	When successful, the interface is exposed on the object and the interface pointer can be obtained by `XAObjectItf::GetInterface()`. This method can be used on a Suspended dynamic interface after reception of a resources available event, `XA_DYNAMIC_ITF_EVENT_RESOURCE_AVAILABLE`.		
See also	None		

RegisterCallback

```
XAresult (*RegisterCallback) (
    XADynamicInterfaceManagementItf self,
    xaDynamicInterfaceManagementCallback callback,
    void * pContext
);
```

Description	Registers a callback on the object that executes when a runtime error, termination of an asynchronous call or change in a dynamic interface's resources occurs.		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the result callback. If `NULL`, the callback is disabled.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		
See also	`xaDynamicInterfaceManagementCallback()`		

8.11 XADynamicSourceItf

Description

This interface is deprecated. Use XADynamicSourceSinkChangeItf [see section 8.12] instead.

This interface exposes a control for changing the data source of the object during the life-time of the object.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_DYNAMICSOURCE;

struct XADynamicSourceItf_;
typedef const struct XADynamicSourceItf_ * const * XADynamicSourceItf;

struct XADynamicSourceItf_ {
    XAresult (*SetSource) (
        XADynamicSourceItf self,
        const XADataSource * pDataSource
    );
};
```

Interface ID

9fc1bce0-de23-11df-8cb0-0002a5d5c51b

Defaults

The data source that was set on object creation.

Methods

SetSource			
XAresult (*SetSource) (**XADynamicSourceItf self,** **const XADataSource * pDataSource** **);**			
Description	Sets the data source for the object.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference
	`pDataSource`	[in]	Pointer to the structure specifying the media data source (such as a container file). Must be non-`NULL`. In the case of a Metadata Extractor object, only local data sources are mandated to be supported.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`
	`XA_RESULT_IO_ERROR`
	`XA_RESULT_CONTENT_CORRUPTED`
	`XA_RESULT_CONTENT_UNSUPPORTED`
	`XA_RESULT_CONTENT_NOT_FOUND`
	`XA_RESULT_PERMISSION_DENIED`		

Comments	Setting a source for a Metadata Extractor object will reset its `XAMetadataExtractionItf` and `XAMetadataTraversalItf` interfaces to point to the new source and reset those interfaces to their initial values. Setting of the new source shall be accepted in any player object state. The playback of the new source shall start from the beginning of the content. The player object shall maintain the same player object state upon accepting the new source. For example, if the player object is currently in XA_PLAYSTATE_PLAYING state, it shall maintain the XA_PLAYSTATE_PLAYING state.		
See also	None		

8.12 XADynamicSourceSinkChangeItf

Description

This interface exposes a control for changing the specified data source or data sink of the object during the life-time of the object. This interface is optional on the MediaPlayer and MediaRecorder objects and implicit on the Metadata Extractor object. . .

Prototype

```
XA_API extern const XAInterfaceID XA_IID_DYNAMICSOURCESINKCHANGE;

struct XADynamicSourceSinkChangeItf_;
typedef const struct XADynamicSourceSinkChangeItf_ * const *
XADynamicSourceSinkChangeItf;

struct XADynamicSourceSinkChangeItf_ {
    XAresult (*ChangeSource) (
        XADynamicSourceSinkChangeItf self,
        const XADataSource * pExistingDataSource,
        const XADataSource * pNewDataSource,
        XAboolean async
    );
    XAresult (*ChangeSink) (
        XADynamicSourceSinkChangeItf self,
        const XADataSink * pExistingDataSink,
        const XADataSink * pNewDataSink,
        XAboolean async
    );
    XAresult (*RegisterSourceChangeCallback) (
        XADynamicSourceSinkChangeItf self,
        xaSourceChangeCallback callback,
        void * pContext
    );
    XAresult (*RegisterSinkChangeCallback) (
        XADynamicSourceSinkChangeItf self,
        xaSinkChangeCallback callback,
        void * pContext
    );
};
```

Interface ID

069a2b20-c432-11df-851a-0800200c9a66

Defaults

None.

Callbacks

xaSourceChangeCallback

```
typedef void (XAAPIENTRY *xaSourceChangeCallback) (
   XASourceSinkChangeItf caller,
   void * pContext,
   XAuint32 resultCode,
   const XADataSource * pExistingDataSource,
   const XADataSource * pNewDataSource
);
```

Description	Callback that executes whenever the ChangeSource method finishes execution.		
Pre-conditions	None.		
Parameters	caller	[in]	Interface instantiation on which the callback was registered
	pContext	[in]	User context data that is supplied when the callback is registered.
	resultCode	[in]	Result code indicating the status of the source change. Possible result codes are the following: XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID XA_RESULT_MEMORY_FAILURE XA_RESULT_IO_ERROR XA_RESULT_CONTENT_CORRUPTED XA_RESULT_CONTENT_UNSUPPORTED XA_RESULT_CONTENT_NOT_FOUND XA_RESULT_PERMISSION_DENIED XA_RESULT_FORMATS_INCOMPATIBLE
	pExistingDataSource	[in]	Pointer to the current data source passed in the ChangeSource method.
	pNewDataSource	[in]	Pointer to the new data source passed in the ChangeSource method.
Comments	The resultCode indicates whether the source change requested by ChangeSource actually succeeded or not. XA_RESULT_SOURCE_SINK_INCOMPATIBLE is returned if the application tries to connect two media objects with incompatible media formats and/or locator types. XA_RESULT_PERMISSION_DENIED is returned if the source to be connected has protected content and the media object does not have the requisite permissions to process it. The pExistingDataSource and pNewDataSource pointers are exactly the same pointers passed in the ChangeSource method in		

xaSourceChangeCallback	
	the XADynamicSourceSinkChangeItf. As such, no implementation memory space is being exposed by this callback. It is worth noting that: • Two successive calls to ChangeSource asking for the same change would result in only one callback. That is, the second call to ChangeSource is ignored. • Two successive calls to ChangeSource asking for different changes will result in two callbacks in the correct order.
See also	RegisterSourceChangeCallback, ChangeSource

xaSinkChangeCallback			
typedef void (XAAPIENTRY *xaSinkChangeCallback) (XASourceSinkChangeItf caller, void * pContext, XAuint32 resultCode, const XADataSink * pExistingDataSink, const XADataSink * pNewDataSink);			
Description	Callback that executes whenever the ChangeSink method finishes execution.		
Pre-conditions	None.		
Parameters	caller	[in]	Interface instantiation on which the callback was registered
	pContext	[in]	User context data that is supplied when the callback is registered.
	resultCode	[in]	Result code indicating the status of the Sink change. Possible result codes are the following: XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID XA_RESULT_MEMORY_FAILURE XA_RESULT_IO_ERROR XA_RESULT_CONTENT_CORRUPTED XA_RESULT_CONTENT_UNSUPPORTED XA_RESULT_CONTENT_NOT_FOUND XA_RESULT_PERMISSION_DENIED XA_RESULT_FORMATS_INCOMPATIBLE
	pExistingDataSink	[in]	Pointer to the current data Sink passed in the ChangeSink method.
	pNewDataSink	[in]	Pointer to the new data Sink passed in the ChangeSink method.
Comments	The resultCode indicates whether the Sink change requested by ChangeSink actually succeeded or not. XA_RESULT_SOURCE_SINK_INCOMPATIBLE is returned if the application tries to connect two media objects with incompatible media formats. XA_RESULT_PERMISSION_DENIED is returned if the sink to be connected does		

xaSinkChangeCallback	
	not have the requisite permissions to handle the protected media content output by the media object. The pExistingDataSink and pNewDataSink pointers are exactly the same pointers passed in the ChangeSink method in the XADynamicSourceSinkChangeItf. As such, no implementation memory space is being exposed by this callback. It is worth noting that: • Two successive calls to ChangeSink asking for the same change would result in only one callback. That is, the second call to ChangeSink is ignored. • Two successive calls to ChangeSink asking for different changes will result in two callbacks in the correct order.
See also	RegisterSinkChangeCallback

Methods

ChangeSource			
XAresult (*ChangeSource) (XADynamicSourceSinkChangeItf self, const XADataSource * pExistingDataSource, const XADataSource * pNewDataSource, XAboolean async);			
Description	Changes the specified data source of the media object.		
Parameters	self	[in]	Interface self-reference
	pExistingDataSource	[in]	Pointer to the structure specifying the existing media data source. Must be non-NULL.
	pNewDataSource	[in]	Pointer to the structure specifying the new media data source. Must be non-NULL. In the case of a Metadata Extractor object, only local data sources are mandated to be supported.
	async	[in]	If XA_BOOLEAN_FALSE, the method will block until termination. Otherwise, the method will return XA_RESULT_SUCCESS, and will be executed asynchronously. On termination, the {xa|sl}SourceChangeCallback() will be invoked, if registered, The resultCode parameter of the callback will contain the result code of the method. However, if the implementation is unable to initiate the asynchronous call XA_RESULT_RESOURCE_ERROR will be returned.

ChangeSource			
Return value	The following return values are possible. XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID XA_RESULT_MEMORY_FAILURE XA_RESULT_IO_ERROR XA_RESULT_CONTENT_CORRUPTED XA_RESULT_CONTENT_UNSUPPORTED XA_RESULT_CONTENT_NOT_FOUND XA_RESULT_PERMISSION_DENIED XA_RESULT_FORMATS_INCOMPATIBLE If invoked as an asynchronous method, it returns XA_RESULT_SUCCESS, and the resultCode parameter of the callback will contain the result of this method's execution.		
Comments	Setting of the new source shall be accepted in any media object state. • For {media|audio} player objects, playback of the new source will start from the beginning of the content. • For {media|audio} recorder objects, the recording of the new source will start from the beginning of the content. The media object shall maintain the same media object state upon accepting the new source. For example, if the player object is currently in XA_PLAYSTATE_PLAYING state, it shall maintain the XA_PLAYSTATE_PLAYING state. This method is asynchronous. The application would need to register for the xaSourceChangeCallback to get notification of the outcome of the source change.		
See also	xaSourceChangeCallback, RegisterSourceChangeCallback		

ChangeSink			
XAresult (*ChangeSink) (XADynamicSourceSinkChangeItf self, const XADataSink * pExistingDataSink, const XADataSink * pNewDataSink, XAboolean async);			
Description	Changes the specified data sink of the media object.		
Parameters	self	[in]	Interface self-reference
	pExistingDataSink	[in]	Pointer to the structure specifying the existing media data sink. Must be non-NULL.
	pNewDataSink	[in]	Pointer to the structure specifying the new media data sink. Must be non-NULL.
	async	[in]	If XA_BOOLEAN_FALSE, the method will block until termination. Otherwise, the method will return XA_RESULT_SUCCESS, and will be

ChangeSink			
			executed asynchronously. On termination, the xaSinkChangeCallback() will be invoked, if registered, The resultCode parameter of the callback will contain the result code of the method. However, if the implementation is unable to initiate the asynchronous call XA_RESULT_RESOURCE_ERROR will be returned.
Return value	The following return values are possible. XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID XA_RESULT_MEMORY_FAILURE XA_RESULT_IO_ERROR XA_RESULT_CONTENT_CORRUPTED XA_RESULT_CONTENT_UNSUPPORTED XA_RESULT_CONTENT_NOT_FOUND XA_RESULT_PERMISSION_DENIED XA_RESULT_FORMATS_INCOMPATIBLE If invoked as an asynchronous method, it returns XA_RESULT_SUCCESS, and the resultCode parameter of the callback will contain the result of this method's execution.		
Comments	Setting of the new sink shall be accepted in any media object state. • For {media|audio} player objects, the playback of the content to the new sink will continue from the current position in the content. • For {media|audio| recorder objects, the recording of the content to the new sink will continue from the current position in the content. The media object shall maintain the same media object state upon accepting the new sink. For example, if the player object is currently in XA_PLAYSTATE_PLAYING state, it shall maintain the XA_PLAYSTATE_PLAYING state. This method is asynchronous. The application would need to register for the xaSinkChangeCallback to get notification of the outcome of the sink change.		
See also	xaSinkChangeCallback, RegisterSinkChangeCallback		

RegisterSourceChangeCallback			
XAresult (*RegisterSourceChangeCallback) (XADynamicSourceSinkChangeItf self, xaSourceChangeCallback callback, void * pContext);			
Description	Sets or clears xaSourceChangeCallback		
Pre-conditions	None.		
Parameters	self	[in]	Interface self-reference
	callback	[in]	Pointer to the callback function to be called when the ChangeSource method finishes execution.

RegisterSourceChangeCallback			
	pContext	[in]	User context data that is to be returned as part of the callback method.
Return value	The return value can be one of the following: XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID		
Comments	None.		
See also	xaSourceChangeCallback		

RegisterSinkChangeCallback			
XAresult (*RegisterSinkChangeCallback) (XADynamicSourceSinkChangeItf self, xaSinkChangeCallback callback, void * pContext);			
Description	Sets or clears xaSinkChangeCallback		
Pre-conditions	None.		
Parameters	self	[in]	Interface self-reference
	callback	[in]	Pointer to the callback function to be called when the ChangeSink method finishes execution.
	pContext	[in]	User context data that is to be returned as part of the callback method.
Return value	The return value can be one of the following: XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID		
Comments	None.		
See also	xaSinkChangeCallback		

8.13 XAEngineItf

Description

This interface exposes creation methods of all the OpenMAX AL object types. See Appendix D: for examples using this interface.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_ENGINE;

struct XAEngineItf_;
typedef const struct XAEngineItf_ * const * XAEngineItf;

struct XAEngineItf_ {
    XAresult (*CreateCameraDevice) (
        XAEngineItf self,
        XAObjectItf * pDevice,
        XAuint32 deviceID,
        XAuint32 numInterfaces,
        const XAInterfaceID * pInterfaceIds,
        const XAboolean * pInterfaceRequired
    );
    XAresult (*CreateRadioDevice) (
        XAEngineItf self,
        XAObjectItf * pDevice,
        XAuint32 numInterfaces,
        const XAInterfaceID * pInterfaceIds,
        const XAboolean * pInterfaceRequired
    );
    XAresult (*CreateLEDDevice) (
        XAEngineItf self,
        XAObjectItf * pDevice,
        XAuint32 deviceID,
        XAuint32 numInterfaces,
        const XAInterfaceID * pInterfaceIds,
        const XAboolean * pInterfaceRequired
    );
     XAresult (*CreateVibraDevice) (
        XAEngineItf self,
        XAObjectItf * pDevice,
        XAuint32 deviceID,
        XAuint32 numInterfaces,
        const XAInterfaceID * pInterfaceIds,
        const XAboolean * pInterfaceRequired
    );
```

```
XAresult (*CreateMediaPlayer) (
    XAEngineItf self,
    XAObjectItf * pPlayer,
    const XADataSource * pDataSrc,
    const XADataSource * pBankSrc,
    const XADataSink * pAudioSnk,
    const XADataSink * pImageVideoSnk,
    const XADataSink * pVibra,
    const XADataSink * pLEDArray,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
XAresult (*CreateMediaRecorder) (
    XAEngineItf self,
    XAObjectItf * pRecorder,
    const XADataSource * pAudioSrc,
    const XADataSource * pImageVideoSrc,
    const XADataSink * pDataSnk,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
XAresult (*CreateOutputMix) (
    XAEngineItf self,
    XAObjectItf * pMix,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
XAresult (*CreateMetadataExtractor) (
    XAEngineItf self,
    XAObjectItf * pMetadataExtractor,
    const XADataSource * pDataSource,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
XAResult (*CreateExtensionObject) (
    XAEngineItf self,
    XAObjectItf * pObject,
    void * pParameters,
    XAuint32 objectID,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
XAresult (*GetImplementationInfo) (
    XAEngineItf self,
    XAuint32 * pMajor,
    XAuint32 * pMinor,
    XAuint32 * pStep,
    Xauint32 nImplementationTextSize,
    XAchar * pImplementationText
);
```

```
    XAresult (*QuerySupportedProfiles) (
        XAEngineItf self,
        XAint16 * pProfilesSupported
    );
    XAresult (*QueryNumSupportedInterfaces) (
        XAEngineItf self,
        XAuint32 objectID,
        XAuint32 * pNumSupportedInterfaces
    );
    XAresult (*QuerySupportedInterfaces) (
        XAEngineItf self,
        XAuint32 objectID,
        XAuint32 index,
        XAInterfaceID * pInterfaceId
    );
    XAresult (*QueryNumSupportedExtensions) (
        XAEngineItf self,
        XAuint32 * pNumExtensions
    );
    XAresult (*QuerySupportedExtension) (
        XAEngineItf self,
        XAuint32 index,
        XAchar * pExtensionName,
        XAuint16 * pNameLength
    );
    XAresult (*IsExtensionSupported) (
        XAEngineItf self,
        const XAchar * pExtensionName,
        XAboolean * pSupported
    );
    XAresult (*QueryLEDCapabilities) (
        XAEngineItf self,
        XAuint32 * pIndex,
        XAuint32 * pLEDDeviceID,
        XALEDDescriptor * pDescriptor
    );
    XAresult (*QueryVibraCapabilities) (
        XAEngineItf self,
        XAuint32 * pIndex,
        XAuint32 * pVibraDeviceID,
        XAVibraDescriptor * pDescriptor
    );
};
```

Interface ID

de9000a0-d6dd-11df-be24-0002a5d5c51b

Defaults

None.

Methods

CreateCameraDevice			
XAresult (*CreateCameraDevice) (**XAEngineItf self,** **XAObjectItf * pDevice,** **XAuint32 deviceID,** **XAuint32 numInterfaces,** **const XAInterfaceID * pInterfaceIds,** **const XAboolean * pInterfaceRequired** **);**			
Description	Creates a camera device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pDevice`	[out]	Newly-created camera device object.
	`deviceID`	[in]	ID of the camera device.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
	`pInterfaceIds`	[in]	Array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
	`pInterfaceRequired`	[in]	Array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`		

Comments	If the engine fails to create the object due to lack of memory or resources it will return the `XA_RESULT_MEMORY_FAILURE` or the `XA_RESULT_RESOURCE_ERROR` error, respectively.		
See also	`XACameraCapabilitiesItf` (see section 8.7) to determine the capabilities of the camera device. Camera I/O device object (see section 7.1).		

CreateRadioDevice			
XAresult (*CreateRadioDevice) (**XAEngineItf self,** **XAObjectItf * pDevice,** **XAuint32 numInterfaces,** **const XAInterfaceID * pInterfaceIds,** **const XAboolean * pInterfaceRequired** **);**			
Description	Creates a radio device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pDevice`	[out]	Newly-created radio device object.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
	`pInterfaceIds`	[in]	Array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
	`pInterfaceRequired`	[in]	Array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`		

Comments	If the engine fails to create the object due to lack of memory or resources it will return the `XA_RESULT_MEMORY_FAILURE` or the `XA_RESULT_RESOURCE_ERROR` error, respectively.		
See also	Radio I/O device object (see section 7.8).		

CreateLEDDevice

```
XAresult (*CreateLEDDevice) (
    XAEngineItf self,
    XAObjectItf * pDevice,
    XAuint32 deviceID,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
```

Description	Creates an LED device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pDevice`	[out]	Newly-created LED device object.
	`deviceID`	[in]	ID of the LED device.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
	`pInterfaceIds`	[in]	Array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
	`pInterfaceRequired`	[in]	Array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`		

Comments	If the engine fails to create the object due to lack of memory or resources it will return the `XA_RESULT_MEMORY_FAILURE` or the `XA_RESULT_RESOURCE_ERROR` error, respectively.		
See also	`XAEngineItf` (see section 8.13) to determine the capabilities of the LED device. LED array I/O device object (see section 7.3).		

CreateVibraDevice

```
XAresult (*CreateVibraDevice) (
    XAEngineItf self,
    XAObjectItf * pDevice,
    XAuint32 deviceID,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
```

Description	Creates a vibrator device.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pDevice`	[out]	Newly-created vibrator device object.
	`deviceID`	[in]	ID of the vibrator device.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
	`pInterfaceIds`	[in]	Array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
	`pInterfaceRequired`	[in]	Array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`		

Comments	If the engine fails to create the object due to lack of memory or resources it will return the `XA_RESULT_MEMORY_FAILURE` or the `XA_RESULT_RESOURCE_ERROR` error, respectively.		
See also	`XAEngineItf` (see section 8.13) to determine the capabilities of the LED device. Vibra I/O device object (see section 7.9).		

CreateMediaPlayer

```
XAresult (*CreateMediaPlayer) (
    XAEngineItf self,
    XAObjectItf * pPlayer,
    const XADataSource * pDataSrc,
    const XADataSource * pBankSrc,
    const XADataSink * pAudioSnk,
    const XADataSink * pImageVideoSnk,
    const XADataSink * pVibra,
    const XADataSink * pLEDArray,
    const XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
```

Description	Creates a media player object.		
Pre-conditions	If data source's or data sink's locator is an object (e.g. camera, radio or output mix) this object must be in the realized state.		
Parameters	`self`	[in]	Interface self-reference.
	`pPlayer`	[out]	Newly-created media player object.
	`pDataSrc`	[in]	Pointer to the structure specifying the data source (such as a container file). For MIDI, the data source must be a Mobile XMF or SP-MIDI file reference.
	`pBankSrc`	[in]	Pointer to the structure specifying the instrument bank in Mobile DLS format. This is an optional parameter. If `NULL` the default bank of instruments definitions is used. This parameter is ignored for non-MIDI data sources.
	`pAudioSnk`	[in]	Pointer to the structure specifying the audio data sink (such as an audio output device). This field may be `NULL` (such as when the data does not contain audio).
	`pImageVideoSnk`	[in]	Pointer to the structure specifying the image/video data sink (such as a native window handle). This field may be `NULL` (such as when the data does not contain video or an image).
	`pVibra`	[in]	Pointer to the structure specifying the Vibra I/O device to which the media player should send vibration data. If `NULL`, no Vibra I/O devices will be controlled. Vibra I/O devices as data sinks may not be supported for non-MIDI media.
	`pLEDArray`	[in]	Pointer to the structure specifying the LED array I/O device to which the media player should send LED array data. If `NULL`, no LED array I/O devices will be controlled. LED array I/O devices as data sinks may not be supported for non-MIDI media.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).

CreateMediaPlayer			
	pInterfaceIds	[in]	Array of numInterfaces interface IDs, which the object should support. This parameter is ignored if numInterfaces is zero.
	pInterfaceRequired	[in]	Array of numInterfaces flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code XA_RESULT_FEATURE_UNSUPPORTED will be then returned. This parameter is ignored if numInterfaces is zero.
Return value	The return value can be one of the following: XA_RESULT_SUCCESS XA_RESULT_PRECONDITIONS_VIOLATED XA_RESULT_PARAMETER_INVALID XA_RESULT_MEMORY_FAILURE		
Comments	If the engine fails to create the object due to lack of memory or resources it will return the XA_RESULT_MEMORY_FAILURE or the XA_RESULT_RESOURCE_ERROR error, respectively.		
See also	Media Player Object (see section 7.4).		

CreateMediaRecorder

```
XAresult (*CreateMediaRecorder) (
    XAEngineItf self,
    XAObjectItf * pRecorder,
    const XADataSource * pAudioSrc,
    const XADataSource * pImageVideoSrc,
    const XADataSink * pDataSnk,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
```

Description	Creates a media recorder.		
Pre-conditions	If data source's locator is an object (e.g. camera or radio) this object must be in the realized state.		
Parameters	`self`	[in]	Interface self-reference.
	`pRecorder`	[out]	Newly-created media recorder object.
	`pAudioSrc`	[in]	Pointer to the structure specifying the audio data source (such as a microphone device). If this field is `NULL` then no audio source is specified and the recorder only captures video or image data.
	`pImageVideoSrc`	[in]	Pointer to the structure specifying the video data source (such as a camera device). If this field is `NULL` then no image/video source is specified and the recorder only captures audio data.
	`pDataSnk`	[in]	Pointer to the structure specifying the audio/video data sink (such as a container file). This parameter is ignored if `XASnapshotItf` (and not `XARecordItf`) is used since `XASnapshotItf::InitiateSnapshot()` is used to define the output of the captured image(s). If `pDataSnk` is a URI (say a file) that already exists, it will be overwritten.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
	`pInterfaceIds`	[in]	Array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
	`pInterfaceRequired`	[in]	Array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

CreateMediaRecorder	

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`
	`XA_RESULT_READONLY`

Comments	If the engine fails to create the object due to lack of memory or resources it will return the `XA_RESULT_MEMORY_FAILURE` or the `XA_RESULT_RESOURCE_ERROR` error, respectively. A value of XA_RESULT_READONLY will be returned when pDataSnk is unable to be written to due to read-only file status.
See also	Media Recorder Object (see section 7.5).

CreateOutputMix			
XAresult (*CreateOutputMix) (**XAEngineItf self, XAObjectItf * pMix,** **XAuint32 numInterfaces,** **const XAInterfaceID * pInterfaceIds,** **const XAboolean * pInterfaceRequired** **);**			
Description	Creates an output mix.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMix`	[out]	Newly-created output mix object.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
	`pInterfaceIds`	[in]	Array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
	`pInterfaceRequired`	[in]	Array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`		

Comments	If the engine fails to create the object due to lack of memory or resources it will return the `XA_RESULT_MEMORY_FAILURE` or the `XA_RESULT_RESOURCE_ERROR` error, respectively.		
See also	Output Mix Object (see section 7.7).		

CreateMetadataExtractor

```
XAresult (*CreateMetadataExtractor) (
    XAEngineItf self,
    XAObjectItf * pMetadataExtractor,
    const XADataSource * pDataSource,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
```

Description	Creates a Metadata Extractor object.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMetadataExtractor`	[out]	Newly created metadata extractor object.
	`pDataSource`	[in]	Pointer to the structure specifying the media data source (such as a media file). Only local data sources are mandated to be supported. Must be non-`NULL`.
	`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
	`pInterfaceIds`	[in]	Array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
	`pInterfaceRequired`	[in]	Array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`
	`XA_RESULT_IO_ERROR`
	`XA_RESULT_CONTENT_CORRUPTED`
	`XA_RESULT_CONTENT_UNSUPPORTED`
	`XA_RESULT_CONTENT_NOT_FOUND`
	`XA_RESULT_PERMISSION_DENIED`		

Comments	None		
See also	Metadata Extractor Object (see section 7.6).		

CreateExtensionObject

```
XAresult (*CreateExtensionObject) (
    XAEngineItf self,
    XAObjectItf * pObject,
    void * pParameters,
    XAuint32 objectID,
    XAuint32 numInterfaces,
    const XAInterfaceID * pInterfaceIds,
    const XAboolean * pInterfaceRequired
);
```

Description

Creates an object. This method is used for extension objects defined externally from the specification. Objects defined by the specification must be created by the specific creation methods in the engine interface.

Pre-conditions

As documented by extension.

Parameters

`self`	[in]	Interface self-reference.
`pObject`	[out]	Newly-created object.
`pParameters`	[in]	Pointer to a structure specifying the parameters used for creating the object.
`objectID`	[in]	A valid object ID.
`numInterfaces`	[in]	Number of interfaces that the object is requested to support (not including implicit interfaces).
`pInterfaceIds`	[in]	Array of `numInterfaces` interface IDs, which the object should support. This parameter is ignored if `numInterfaces` is zero.
`pInterfaceRequired`	[in]	Array of `numInterfaces` flags, each specifying whether the respective interface is required on the object or optional. A required interface will fail the creation of the object if it cannot be accommodated and the error code `XA_RESULT_FEATURE_UNSUPPORTED` will be then returned. This parameter is ignored if `numInterfaces` is zero.

Return value

The return value can be one of the following:

`XA_RESULT_SUCCESS`
`XA_RESULT_PRECONDITIONS_VIOLATED`
`XA_RESULT_PARAMETER_INVALID`
`XA_RESULT_MEMORY_FAILURE`
`XA_RESULT_IO_ERROR`
`XA_RESULT_PERMISSION_DENIED`

Comments

If the engine fails to create the object due to lack of memory or resources it will return the `XA_RESULT_MEMORY_FAILURE` or the `XA_RESULT_RESOURCE_ERROR` error, respectively. The `ObjectID` and the data structure pointed to by `pParameters` should be defined by an extension. When `ObjectID` is not valid the method will return `XA_RESULT_FEATURE_UNSUPPORTED.`

CreateExtensionObject	
See also	Section 3.5

GetImplementationInfo			
```XAresult (*GetImplementationInfo) (XAEngineItf self, XAuint32 * pMajor, XAuint32 * pMinor, XAuint32 * pStep, Xauint32 * pImplementationTextSize, XAchar * pImplementationText);```			
Description	Queries the OpenMAX AL implementation information.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMajor`	[out]	Major version number.
	`pMinor`	[out]	Minor version number.
	`pStep`	[out]	Step within the minor version number.
	`pImplementationTextSize`	[in/out]	Size of the implementationText string. As input, specifies the size of the pImplementationText array to be allocated by the application. As output, returns the size of the implementation text.
	`pImplementationText`	[out]	Text describing the implementation including. This text must identify whether the implementation leverages OpenMAX IL partially, fully, or not at all. Beyond this requirement, the actual contents of this string is up to the implementation's discretion.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	For 1.1.0 implementations of OpenMAX AL, this method should return 1, 1, and 0 for the Major, Minor, and Step fields respectively.		
See also	None		

<table>
<tr><th colspan="4">QuerySupportedProfiles</th></tr>
<tr><td colspan="4">XAresult (*QuerySupportedProfiles) (
    XAEngineItf self,
    XAint16 * pProfilesSupported
);</td></tr>
<tr><td>Description</td><td colspan="3">Queries supported profiles of the OpenMAX AL implementation.</td></tr>
<tr><td rowspan="2">Parameters</td><td>Self</td><td>[in]</td><td>Interface self-reference.</td></tr>
<tr><td>pProfilesSupported</td><td>[out]</td><td>Bitmask containing profiles supported, as defined in the XA_PROFILE macros.</td></tr>
<tr><td>Return value</td><td colspan="3">The return value can be one of the following:
XA_RESULT_SUCCESS
XA_RESULT_PARAMETER_INVALID</td></tr>
<tr><td>Comments</td><td colspan="3">Valid values of pProfilesSupported are XA_PROFILES_MEDIA_PLAYER, (XA_PROFILES_MEDIA_PLAYER | XA_PROFILES_MEDIA_PLAYER_RECORDER), (XA_PROFILES_MEDIA_PLAYER | XA_PROFILES_PLUS_MIDI) and (XA_PROFILES_MEDIA_PLAYER | XA_PROFILES_MEDIA_PLAYER_RECORDER | XA_PROFILES_PLUS_MIDI).</td></tr>
<tr><td>See also</td><td colspan="3">None</td></tr>
</table>

QueryNumSupportedInterfaces			
XAresult (*QueryNumSupportedInterfaces) (**XAEngineItf self,** **XAuint32 objectID,** **XAuint32 * pNumSupportedInterfaces** **);**			
Description	Queries the number of supported interfaces available.		
Parameters	`self`	[in]	Interface self-reference.
	`objectID`	[in]	ID of the object being queried. Refer to `XA_OBJECTID` type. If the engine does not support the identified object this method will return `XA_RESULT_FEATURE_UNSUPPORTED`.
	`pNumSupportedInterfaces`	[out]	Identifies the number of supported interfaces available.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The number of supported interfaces will include both mandated and optional interfaces available for the object. This method can be used to determine whether or not an object is supported by an implementation by examining the return value.		
See also	`QuerySupportedInteraces()`		

QuerySupportedInterfaces

```
XAresult (*QuerySupportedInterfaces) (
    XAEngineItf self,
    XAuint32 objectID,
    XAuint32 index,
    XAInterfaceID * pInterfaceId
);
```

Description	Queries the supported interfaces.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`objectID`	[in]	ID of the object being queried. Refer to `XA_OBJECTID` type. If the engine does not support the identified object this method will return `XA_RESULT_FEATURE_UNSUPPORTED`.
	`index`	[in]	Incrementing index used to enumerate available interfaces. Supported index range is 0 to N-1, where N is the number of supported interfaces.
	`pInterfaceId`	[out]	Identifies the supported interface.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The number of supported interfaces will include both mandated and optional interfaces available for the object.		
See also	`QueryNumSupportedInterfaces()`		

QueryNumSupportedExtensions

```
XAresult (*QueryNumSupportedExtensions) (
   XAEngineItf self,
   XAuint32 * pNumExtensions
);
```

Description	Queries the number of supported extensions.		
Parameters	`self`	[in]	Interface self-reference.
	`pNumExtensions`	[out]	Identifies the number of supported extensions by this implementation.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The number of supported extensions will include both standardized extensions listed in Khronos registry and vendor-specific extensions.		
See also	`QuerySupportedExtensions()`		

QuerySupportedExtension

```
XAresult (*QuerySupportedExtension) (
    XAEngineItf self,
    XAuint32 index,
    XAchar * pExtensionName,
    XAuint16 * pNameLength
);
```

Description	Gets the name of the extension supported by the implementation based on the given index.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	The index of the extension. Must be [0, numExtensions-1].
	`pExtensionName`	[out]	The name of the supported extension, as defined in the Khronos registry (http://www.khronos.org/registry/) or in vendor-specific documentation. The length of the needed char array should be first figured out from `pNameLength` out parameter by calling this method with `pExtensionName` as null.
	`pNameLength`	[in/out]	As an output, specifies the length of the name including the terminating `NULL`. As an input, specifies the length of the given `pExtensionName` char array (ignored if `pExtensionName` is `NULL`).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	If the given length is smaller than the needed size XA_RESULT_BUFFER_INSUFFICIENT is returned and only data of the given size will be written; however, no invalid strings are written. That is, the null-terminator always exists and multibyte characters are not cut in the middle.		
See Also	`QueryNumSupportedExtensions()`, `IsExtensionSupported()`		

IsExtensionSupported			
XAresult (*IsExtensionSupported) (**XAEngineItf self,** **const XAchar * pExtensionName,** **XAboolean * pSupported** **);**			
Description	Queries if the given extension is supported by the implementation.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pExtensionName`	[in]	The name of an extension, as defined in the Khronos registry (http://www.khronos.org/registry/) or in vendor-specific documentation. Must be null-terminated.
	`pSupported`	[out]	`XA_BOOLEAN_TRUE` if the given extension is supported; `XA_BOOLEAN_FALSE` if it is not supported.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	This is an alternative method to be used instead of `QueryNumSupportedExtensions()` and `QuerySupportedExtension()` to query the availability of just one known extension.		
See Also	None		

QueryLEDCapabilities			
XAresult (*QueryLEDCapabilities) (**XAEngineItf self,** **XAuint32 * pIndex,** **XAuint32 * pLEDDeviceID,** **XALEDDescriptor * pDescriptor** **);**			
Description	Queries the LED device for its capabilities.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pIndex`	[in/out]	As an input, specifies which LED array device to obtain the capabilities of, the supported range is [0, *n*), where *n* is the number of LED array devices available (ignored if `pDescriptor` is NULL). As an output, specifies the number of LED array devices available in the system. Returns 0 if no LED array devices are available.
	`pLEDDeviceId`	[in/out]	If `pIndex` is non-NULL then returns the LED array device ID corresponding to LED array device `pIndex`. If pIndex is NULL then, as an input, specifies which LED array device to obtain the capabilities of *(*`XA_DEFAULTDEVICEID_LED` can be used to determine the default LED array's capabilities).
	`pDescriptor`	[out]	Structure defining the capabilities of the LED array device.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	An application can determine the number of LED array devices by calling this method with `pDescriptor` set to NULL and examining pIndex. The application can then determine the capabilties of all the LED array devices by calling this method multiple times with `pIndex` pointing to each different index from 0 up to one less than the number of LED array devices. An LED array device is selected using the `CreateLEDDevice()` method.		
See also	XA_DEFAULTDEVICEID_LED [see section 9.2.26]		

QueryVibraCapabilities			
XAresult (*QueryVibraCapabilities) (**XAEngineItf self,** **XAuint32 * pIndex,** **XAuint32 * pVibraDeviceID,** **XAVibraDescriptor * pDescriptor** **);**			
Description	Queries the vibration device for its capabilities.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pIndex`	[in/out]	As an input, specifies which vibration device to obtain the capabilities of, the supported range is [0, *n*), where *n* is the number of vibration devices available (ignored if `pDescriptor` is NULL). As an output, specifies the number of vibration devices available in the system. Returns 0 if no vibration devices are available.
	`pVibraDeviceId`	[in/out]	If `pIndex` is non-NULL then returns the vibration device ID corresponding to vibration device `pIndex`. If pIndex is NULL then, as an input, specifies which vibration device to obtain the capabilities of *(*`XA_DEFAULTDEVICEID_VIBRA` can be used to determine the default vibration device's capabilities).
	`pDescriptor`	[out]	Structure defining the capabilities of the vibration device.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	An application can determine the number of vibration devices by calling this method with `pDescriptor` set to NULL and examining pIndex. The application can then determine the capabilties of all the vibration devices by calling this method multiple times with `pIndex` pointing to each different indexes from 0 up to one less than the number of vibration devices. A vibration device is selected using the `CreateVibraDevice()` method.		
See also	XA_DEFAULTDEVICEID_VIBRA [see section 9.2.26]		

8.14 XAEqualizerItf

Description

XAEqualizerItf is an interface for manipulating the equalization settings of a media object. The equalizer (EQ) can be set up in two different ways: by setting individual frequency bands, or by using predefined presets.

The preset settings can be directly taken into use with the method UsePreset(). The current preset can be queried with the method GetPreset(). If none of the presets is set, XA_EQUALIZER_UNDEFINED will be returned. XA_EQUALIZER_UNDEFINED will also be returned when a preset has been set, but the equalizer settings have been altered later with SetBandLevel(). Presets have names that can be used in the user interface.

There are methods for getting and setting individual EQ-band gains (SetBandLevel() and GetBandLevel()), methods for querying the number of the EQ-bands available (GetNumberOfBands()) and methods for querying their center frequencies (GetCenterFreq()).

The gains in this interface are defined in millibels (hundredths of a decibel), but it has to be understood that many devices contain a Dynamic Range Control (DRC) system that will affect the actual effect and therefore the value in millibels will affect as a guideline rather than as a strict rule.

This interface affects different parts of the audio processing chain, depending on which object the interface is exposed. If this interface is exposed on an Output Mix object, the effect is applied to the output mix. If this interface is exposed on a Player object, it is applied to the Player's output only.

This interface is supported on the Output Mix (see section 7.7) object. See section D.1 for an example using this interface.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_EQUALIZER;

struct XAEqualizerItf_;
typedef const struct XAEqualizerItf_ * const * XAEqualizerItf;

struct XAEqualizerItf_ {
    XAresult (*SetEnabled) (
        XAEqualizerItf self,
        XAboolean enabled
    );
    XAresult (*IsEnabled) (
        XAEqualizerItf self,
        XAboolean * pEnabled
    );
    XAresult (*GetNumberOfBands) (
        XAEqualizerItf self,
        XAuint16 * pNumBands
    );
    XAresult (*GetBandLevelRange) (
        XAEqualizerItf self,
        XAmillibel * pMin,
```

```
        XAmillibel * pMax
    );
    XAresult (*SetBandLevel) (
        XAEqualizerItf self,
        XAuint16 band,
        XAmillibel level
    );
    XAresult (*GetBandLevel) (
        XAEqualizerItf self,
        XAuint16 band,
        XAmillibel * pLevel
    );
    XAresult (*GetCenterFreq) (
        XAEqualizerItf self,
        XAuint16 band,
        XAmilliHertz * pCenter
    );
    XAresult (*GetBandFreqRange) (
        XAEqualizerItf self,
        XAuint16 band,
        XAmilliHertz * pMin,
        XAmilliHertz * pMax
    );
    XAresult (*GetBand) (
        XAEqualizerItf self,
        XAmilliHertz frequency,
        XAuint16 * pBand
    );
    XAresult (*GetCurrentPreset) (
        XAEqualizerItf self,
        XAuint16 * pPreset
    );
    XAresult (*UsePreset) (
        XAEqualizerItf self,
        XAuint16 index
    );
    XAresult (*GetNumberOfPresets) (
        XAEqualizerItf self,
        XAuint16 * pNumPresets
    );
    XAresult (*GetPresetName) (
        XAEqualizerItf self,
        XAuint16 index,
        XAuint16 * pSize,
        XAchar * pName
    );
};
```

Interface ID

68847840-d6de-11df-a826-0002a5d5c51b

Defaults

Enabled: false (disabled)

All band levels: 0 mB (flat response curve)

Preset: `XA_EQUALIZER_UNDEFINED` (no preset)

Methods

SetEnabled			
XAresult (*SetEnabled) (**XAEqualizerItf self,** **XAboolean enabled** **);**			
Description	Enables the effect.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`enabled`	[in]	True to turn on the effect; false to switch it off.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_CONTROL_LOST`		

Comments	None		

IsEnabled			
XAresult (*IsEnabled) (**XAEqualizerItf self,** **XAboolean * pEnabled** **);**			
Description	Gets the enabled status of the effect.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pEnabled`	[out]	True if the effect is on, otherwise false. This must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetNumberOfBands

```
XAresult (*GetNumberOfBands) (
    XAEqualizerItf self,
    XAuint16 * pNumBands
);
```

Description	Gets the number of frequency bands that the equalizer supports. A valid equalizer must have at least two bands.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumBands`	[out]	Number of frequency bands that the equalizer supports. This must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetBandLevelRange

```
XAresult (*GetBandLevelRange) (
    XAEqualizerItf self,
    XAmillibel * pMin,
    XAmillibel * pMax
);
```

Description	Returns the minimum and maximum band levels supported.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMin`	[out]	Minimum supported band level in millibels.
	`pMax`	[out]	Maximum supported band level in millibels.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The range returned by GetBandLevelRange must at least include 0mB. The application may pass NULL as one of the [out] parameters to find out only the other one's value.		

SetBandLevel			
XAresult (*SetBandLevel) (XAEqualizerItf self, XAuint16 band, XAmillibel level);			
Description	Sets the given equalizer band to the given gain value.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`band`	[in]	Frequency band that will have the new gain. The numbering of the bands starts from 0 and ends at (number of bands – 1).
	`level`	[in]	New gain in millibels that will be set to the given band. `getBandLevelRange()` will define the maximum and minimum values.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTROL_LOST`		

Comments	None		

GetBandLevel			
XAresult (*GetBandLevel) (XAEqualizerItf self, XAuint16 band, XAmillibel * pLevel);			
Description	Gets the gain set for the given equalizer band.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`band`	[in]	Frequency band whose gain is requested. The numbering of the bands starts from 0 and ends at (number of bands – 1).
	`pLevel`	[out]	Gain in millibels of the given band. This must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetCenterFreq			
XAresult (*GetCenterFreq) (**XAEqualizerItf self,** **XAuint16 band,** **XAmilliHertz * pCenter** **);**			
Description	Gets the center frequency of the given band.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`band`	[in]	Frequency band whose center frequency is requested. The numbering of the bands starts from 0 and ends at (number of bands – 1).
	`pCenter`	[out]	The center frequency in milliHertz. This must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetBandFreqRange

```
XAresult (*GetBandFreqRange) (
    XAEqualizerItf self,
    XAuint16 band,
    XAmilliHertz * pMin,
    XAmilliHertz * pMax
);
```

Description	Gets the frequency range of the given frequency band.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`band`	[in]	Frequency band whose frequency range is requested. The numbering of the band that can be used with this method starts from 0 and ends at (number of bands – 1).
	`pMin`	[out]	The minimum frequency in milliHertz.
	`pMax`	[out]	The maximum frequency in milliHertz.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The exposed band ranges do not overlap (physically they many times do, but the virtual numbers returned here do not) - this is in order to simplify the applications that want to use this information for graphical representation of the EQ. If shelving filters are used in the lowest and the highest band of the equalizer, the lowest band returns 0 mHz as the minimum frequency and the highest band returns the `XA_MILLIHERTZ_MAX` as the maximum frequency. The application may pass NULL as one of the [out] parameters to find out only the other one's value.		

GetBand

```
XAresult (*GetBand) (
    XAEqualizerItf self,
    XAmilliHertz frequency,
    XAuint16 * pBand
);
```

Description	Gets the band that has the most effect on the given frequency. If no band has an effect on the given frequency, `XA_EQUALIZER_UNDEFINED` is returned.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`frequency`	[in]	Frequency in milliHertz which is to be equalized via the returned band
	`pBand`	[out]	Frequency band that has most effect on the given frequency or `XA_EQUALIZER_UNDEFINED` if no band has an effect on the given frequency. This must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetCurrentPreset

```
XAresult (*GetCurrentPreset) (
    XAEqualizerItf self,
    XAuint16 * pPreset
);
```

Description	Gets the current preset.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pPreset`	[out]	Preset that is set at the moment. If none of the presets are set, `XA_EQUALIZER_UNDEFINED` will be returned. This must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

UsePreset			
XAresult (*UsePreset) (**XAEqualizerItf self,** **XAuint16 index** **);**			
Description	Sets the equalizer according to the given preset.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	New preset that will be taken into use. The valid range is [0, number of presets-1].

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTROL_LOST`		

Comments	None		

GetNumberOfPresets			
XAresult (*GetNumberOfPresets) (**XAEqualizerItf self,** **XAuint16 * pNumPresets** **);**			
Description	Gets the total number of presets the equalizer supports. The presets will have indices [0, number of presets-1].		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumPresets`	[out]	Number of presets the equalizer supports. This must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetPresetName			
XAresult (*GetPresetName) (**XAEqualizerItf self,** **XAuint16 index,** **XAuint16 * pSize,** **XAchar * pName** **);**			
Description	Gets the preset name based on the index.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Index of the preset. The valid range is [0, number of presets-1].
	`pSize`	[in/out]	As an output, pSize will be returned with the character count required to store the name. If pName is not NULL, then as an input, pSize specifies the size of pName. This parameter must not be NULL.
	`pName`	[out]	A non-empty, null terminated string containing the name of the given preset. The character coding is UTF-8.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

8.15 XAImageControlsItf

Description

The image and color controls interface is used to apply adjustments to the associated I/O device. It is realized on a object that supports image or video content. Changes in brightness, contrast and gamma are applied at the beginning of a frame.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_IMAGECONTROLS;

struct XAImageControlsItf_;
typedef const struct XAImageControlsItf_ * const * XAImageControlsItf;

struct XAImageControlsItf_ {
    XAresult (*SetBrightness) (
        XAImageControlsItf self,
        XAuint32 brightness
    );
    XAresult (*GetBrightness) (
        XAImageControlsItf self,
        XAuint32 * pBrightness
    );
    XAresult (*SetContrast) (
        XAImageControlsItf self,
        XAint32 contrast
    );
    XAresult (*GetContrast) (
        XAImageControlsItf self,
        XAint32 * pContrast
    );
    XAresult (*SetGamma) (
        XAImageControlsItf self,
        XApermille gamma
    );
    XAresult (*GetGamma) (
        XAImageControlsItf self,
        XApermille * pGamma
    );
    XAresult (*GetSupportedGammaSettings) (
        XAImageControlsItf self,
        XApermille * pMinValue,
        XApermille * pMaxValue,
        XAuint32 * pNumSettings,
        XApermille * pSettings
    );
};
```

Interface ID

3a449960-d71a-11df-a6f9-0002a5d5c51b

Defaults

The default values are 50 for brightness, 0 for contrast and 1000 for gamma.

Methods

SetBrightness			
XAresult (*SetBrightness) (XAImageControlsItf self, XAuint32 brightness);			
Description	Sets the brightness level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`brightness`	[in]	Defines the brightness level. The value for brightness ranges from 0 to 100, where 0 produces all black pixels and 100 produces all white. `XA_RESULT_PARAMETER_INVALID` is returned if an unsupported brightness level is requested.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	`GetBrightness()`		

GetBrightness			
XAresult (*GetBrightness) (**XAImageControlsItf self,** **XAuint32 * pBrightness** **);**			
Description	Gets the current brightness level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pBrightness`	[out]	Current brightness level. The value for brightness ranges from 0 to 100, where 0 produces all black pixels and 100 produces all white.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	`SetBrightness()`		

SetContrast			
XAresult (*SetContrast) (**XAImageControlsItf self,** **XAint32 contrast** **);**			
Description	Sets the contrast level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`contrast`	[in]	Defines the contrast level. The value for contrast ranges from -100 to 100, where 0 indicates no contrast change. `XA_RESULT_PARAMETER_INVALID` is returned if an unsupported contrast level is requested.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	`GetContrast()`		

GetContrast			
XAresult (*GetContrast) (**XAImageControlsItf self,** **XAint32 * pContrast** **);**			
Description	Gets the current contrast level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pContrast`	[out]	Current contrast level. The value for contrast ranges from -100 to 100, where 0 indicates no contrast change.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	`SetContrast()`		

SetGamma			
XAresult (*SetGamma) (**XAImageControlsItf self,** **XApermille gamma** **);**			
Description	Sets the gamma level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`gamma`	[in]	Defines the gamma level. `XA_RESULT_PARAMETER_INVALID` is returned if an unsupported gamma level is requested.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	`GetGamma()`		

GetGamma			
XAresult (*GetGamma) (XAImageControlsItf self, XApermille * pGamma);			
Description	Gets the current gamma level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pGamma`	[out]	Current gamma level.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	`SetGamma()`		

GetSupportedGammaSettings			
XAresult (*GetSupportedGammaSettings) (**XAImageControlsItf self,** **XApermille * pMinValue,** **XApermille * pMaxValue,** **XAuint32 * pNumSettings,** **XApermille * pSettings** **);**			
Description	This method gets the supported gamma settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMinValue`	[out]	Identifies the minimum gamma setting supported.
	`pMaxValue`	[out]	Identifies the maximum gamma setting supported.
	`pNumSettings`	[in/out]	If `pSettings` is `NULL`, `pNumSettings` returns the number of supported gamma settings. If the available manual settings are continuous from `pMinValue` to `pMaxValue`, `pNumSettings` returns `0`. If `pSettings` is non-`NULL` and a non-continuous range is supported, `pNumSettings` is length of the `pSettings` array.
	`pSettings`	[out]	Returns an array of supported gamma settings. `pSettings` may be `NULL`. The array of values returned must include `pMinValue` and `pMaxValue`..

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If gamma settings are unsupported, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`. The memory for the pSettings array shall be allocated by the OpenMAX AL application.		
See also	None		

8.16 XAImageDecoderCapabilitiesItf

Description

This interface provides methods of querying the image decoding capabilities of the media engine.

This interface provides a means of enumerating all image decoders available on an engine where each an decoderId represents each decoder. It also provides a means to query the capabilities of each decoder.

The set of image decoders supported by the engine does not change during the lifetime of the engine though dynamic resource constraints may limit actual availability when an image decoder is requested.

This interface is a mandated interface of engine objects (see section 7.2).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_IMAGEDECODERCAPABILITIES;

struct XAImageDecoderCapabilitiesItf_;
typedef const struct XAImageDecoderCapabilitiesItf_
    * const * XAImageDecoderCapabilitiesItf;

struct XAImageDecoderCapabilitiesItf_ {
    XAresult (*GetImageDecoderCapabilities) (
        XAImageDecoderCapabilitiesItf self,
        XAuint32 * pDecoderId,
        XAImageCodecDescriptor * pDescriptor
    );
    XAresult (*QueryColorFormats) (
        const XAImageDecoderCapabilitiesItf self,
        XAuint32 * pIndex,
        XAuint32 * pColorFormat
    );
};
```

Interface ID

```
c333e7a0-e616-11dc-a93e-0002a5d5c51b
```

Defaults

Not applicable

Methods

GetImageDecoderCapabilities			
XAresult (*GetImageDecoderCapabilities) (**XAImageDecoderCapabilitiesItf self,** **XAuint32 * pDecoderId,** **XAImageCodecDescriptor * pDescriptor** **);**			
Description	Retrieves image decoder capabilities.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pDecoderId`	[in/out]	If `pDescriptor` is `NULL`, pDecoderId returns the number of image decoders. All implementations must have at least one decoder. If `pDescriptor` is non-`NULL`, pDecoderId is a incrementing value used to enumerate image decoders. Supported index range is 0 to N-1, where N is the number of image decoders.
	`pDescriptor`	[out]	Structure defining the capabilities of the image decoder.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If `XA_IMAGECODEC_RAW` is one of the image codecs supports, `QueryColorFormats()` should be used to determine the color formats supported.		
See also	`QueryColorFormats()`		

QueryColorFormats			
XAresult (*QueryColorFormats) (**const XAImageDecoderCapabilitiesItf self,** **XAuint32 * pIndex,** **XAuint32 * pColorFormat** **);**			
Description	This method is used to query the color formats supported by the image decoder.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pIndex`	[in/out]	If `pColorFormats` is `NULL`, `pIndex` returns the number of color formats supported. Returns 0 if there are no color formats are supported. If `pColorFormats` is non-`NULL`, `pIndex` is an incrementing value used for enumerating the color formats supported. Supported index range is 0 to N-1, where N is the number of color format supports.
	`pColorFormat`	[out]	Pointer to the color format. May be `NULL`. See `XA_COLORFORMAT` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	This method returns color formats associated with the `XA_IMAGECODEC_RAW` codec.		
See also	None		

8.17 XAImageEffectsItf

Description

The image effects interface is used to determine which effects are supported, to enable effects and to disable effects. It is realized on a object that supports image or video content. `QuerySupportedEffects()` or `EnableImageEffect()` can be called to determine if a specific effect is supported. The image effects supported by a media object may not change during the lifetime of the media object.

A platform may allow effects to support optional sets of parameters that control that effect, such as supplying a threshold or strength field. These should be supplied using the configuration extensions interface (see section 8.8).

If XAImageEffectsItf is implemented, it shall support at least one effect.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_IMAGEEFFECTS;

struct XAImageEffectsItf_;
typedef const struct XAImageEffectsItf_ * const * XAImageEffectsItf;

struct XAImageEffectsItf_ {
    XAresult (*QuerySupportedImageEffects) (
        XAImageEffectsItf self,
        XAuint32 index,
        XAuint32 * pImageEffectId
    );
    XAresult (*EnableImageEffect) (
        XAImageEffectsItf self,
        XAuint32 imageEffectID
    );
    XAresult (*DisableImageEffect) (
        XAImageEffectsItf self,
        XAuint32 imageEffectID
    );
    XAresult (*IsImageEffectEnabled) (
        XAImageEffectsItf self,
        XAuint32 imageEffectID,
        XAboolean * pEnabled
    );
};
```

Interface ID

b865bca0-df04-11db-bab9-0002a5d5c51b

Defaults

Initially all effects are disabled.

Methods

QuerySupportedImageEffects			
XAresult (*QuerySupportedImageEffects) (XAImageEffectsItf self, XAuint32 index, XAuint32 * pImageEffectId);			
Description	Queries image effects supported.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Incrementing index used to enumerate available effects. Supported index range is 0 to N-1, where N is the number of effects.
	`pImageEffectId`	[out]	Identifies the supported image effect. Refer to `XA_IMAGEEFFECT` macro (see section 9.2.37).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If `XAImageEffectsItf` is implemented, it shall support at least one effect.. The image effects supported by a media object may not change during the lifetime of the media object.		
See also	`EnableImageEffect()`, `DisableImageEffect()`, `IsImageEffectEnabled()`		

EnableImageEffect			
XAresult (*EnableImageEffect) (XAImageEffectsItf self, XAuint32 imageEffectID);			
Description	Enables an image effect.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`imageEffectId`	[in]	Identifies the supported image effect. Refer to `XA_IMAGEEFFECT` macro (see section 9.2.37).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_RESOURCE_ERROR`		

Comments	Vendor implementations may allow multiple image effects to be enabled simultaneously. If enabling the requested image effect requires disabling a previously enabled image effect, the requested image effect will not be enabled and the `XA_RESULT_RESOURCE_ERROR` will be returned. When multiple image effects are enabled, the order for which these image effects are applied is vendor implementation specific.		
See also	`QuerySupportedImageEffect()`, `DisableImageEffect()`, `IsImageEffectEnabled()`		

DisableImageEffect			
XAresult (*DisableImageEffect) (XAImageEffectsItf self, XAuint32 imageEffectID);			
Description	Disable an image effect.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`imageEffectId`	[in]	Identifies the supported image effect. Refer to `XA_IMAGEEFFECT` macro (see section 9.2.37).
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	If the specified image effect is already disabled, `XA_RESULT_SUCCESS` will be return.		
See also	`QuerySupportedImageEffect()`, `EnableImageEffect()`, `IsImageEffectEnabled()`		

IsImageEffectEnabled			
XAresult (*IsImageEffectEnabled) (**XAImageEffectsItf self,** **XAuint32 imageEffectID,** **XAboolean * pEnabled** **);**			
Description	Checks to see if an image effect is enabled.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`imageEffectId`	[in]	Identifies the supported image effect. Refer to `XA_IMAGEEFFECT` macro (see section 9.2.37).
	`pEnabled`	[out]	Identifies if the image effect is enabled, `XA_BOOLEAN_TRUE` indicates that the effect is enabled and `XA_BOOLEAN_FALSE` indicates that the effect is disabled.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`QuerySupportedImageEffect()`, `DisableImageEffect()`, `EnableImageEffect()`		

8.18 XAImageEncoderItf

Description

This interface is used to set the parameters to be used by an image encoder.

This interface is a mandated interface of Media Recorder objects (see section 7.5).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_IMAGEENCODER;

struct XAImageEncoderItf_;
typedef const struct XAImageEncoderItf_ * const * XAImageEncoderItf;

struct XAImageEncoderItf_ {
    XAresult (*SetImageSettings) (
        XAImageEncoderItf self,
        const XAImageSettings * pSettings
    );
    XAresult (*GetImageSettings) (
        XAImageEncoderItf self,
        XAImageSettings * pSettings
    );
    XAresult (*GetSizeEstimate) (
        XAImageEncoderItf self,
        XAuint32 * pSize
    );
};
```

Interface ID

cd49f140-df04-11db-8888-0002a5d5c51b

Defaults

The default value for image is JPEG, for image width is 640, for image height is 480, and for compression level is 0.

Methods

SetImageSettings			
XAresult (*SetImageSettings) (XAImageEncoderItf self, const XAImageSettings * pSettings);			
Description	Set image encoder settings.		
Pre-conditions	Settings shall be applied prior to initiating a snapshot request – SnapShotItf::InitiateSnaphot		
Parameters	`self`	[in]	Interface self-reference.
	`pSettings`	[in]	Image encoder settings.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	None		
See also	`GetImageSetting()`		

GetImageSettings			
XAresult (*GetImageSettings) (XAImageEncoderItf self, XAImageSettings * pSettings);			
Description	Get image encoder settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pSettings`	[out]	Image encoder settings.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`SetImageSetting()`		

GetSizeEstimate			
XAresult (*GetSizeEstimate) (**XAImageEncoderItf self,** **XAuint32 * pSize** **);**			
Description	Get estimated image size.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pSize`	[out]	Estimated encoding size, in bytes, of the image based on current settings.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`SetImageSettings()`		

8.19 XAImageEncoderCapabilitiesItf

Description

This interface provides methods of querying the image encoding capabilities of the media engine.

This interface provides a means of enumerating all image encoders available on an engine where an encoderId represents each encoder. It also provides a means for querying the capabilities of each encoder.

The set of image encoders supported by the engine does not change during the lifetime of the engine, though dynamic resource constraints may limit actual availability when an image encoder is requested.

This interface is a mandated interface of engine objects (see section 7.2).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_IMAGEENCODERCAPABILITIES;

struct XAImageEncoderCapabilitiesItf_;
typedef const struct XAImageEncoderCapabilitiesItf_
    * const * XAImageEncoderCapabilitiesItf;

struct XAImageEncoderCapabilitiesItf_ {
    XAresult (*GetImageEncoderCapabilities) (
        XAImageEncoderCapabilitiesItf self,
        XAuint32 * pEncoderId,
        XAImageCodecDescriptor * pDescriptor
    );
    XAresult (*QueryColorFormats) (
        const XAImageEncoderCapabilitiesItf self,
        XAuint32 * pIndex,
        XAuint32 * pColorFormat
    );
};
```

Interface ID

c19f0640-e86f-11db-b2d2-0002a5d5c51b

Defaults

Not applicable

Methods

GetImageEncoderCapabilities			
XAresult (*GetImageEncoderCapabilities) (**XAImageEncoderCapabilitiesItf self,** **XAuint32 * pEncoderId,** **XAImageCodecDescriptor * pDescriptor** **);**			
Description	Retrieves image encoder capabilities.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pEncoderId`	[in/out]	If `pDescriptor` is `NULL`, `pEncoderId` returns the number of image encoders. Returns 0 if the engine does not provide any image encoders. If `pDescriptor` is non-`NULL`, `pEncoderId` is a incrementing value used to enumerate image encoders. Supported index range is 0 to N-1, where N is the number of image encoders.
	`pDescriptor`	[out]	Structure defining the capabilities of the image encoder.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If `XA_IMAGECODEC_RAW` is one of the image codecs supports, `QueryColorFormats()` should be used to determine the color formats supported. **PROFILE NOTES** *A Media Player/Recorder profile implementation must support at least one encoder.*		
See also	`QueryColorFormats()`		

QueryColorFormats			
XAresult (*QueryColorFormats) (**const XAImageEncoderCapabilitiesItf self,** **XAuint32 * pIndex,** **XAuint32 * pColorFormat** **);**			
Description	This method is used to query the color formats supported by the image encoder.		
Pre-conditions	None		
Parameters	`Self`	[in]	Interface self-reference.
	`pIndex`	[in/out]	If `pColorFormats` is `NULL`, `pIndex` returns the number of color formats supported. Returns 0 if there are no color formats are supported. If `pColorFormats` is non-`NULL`, `pIndex` is an incrementing value used for enumerating the color formats supported. Supported index range is 0 to N-1, where N is the number of color format supports.
	`pColorFormat`	[out]	Pointer to the color format. May be `NULL`. See `XA_COLORFORMAT` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS – Success.`
	`XA_RESULT_PARAMETER_INVALID – One or more of the parameters passed to the method are invalid.`		

Comments	This method returns color formats associated with the `XA_IMAGECODEC_RAW` codec.		
See also	None		

8.20 XALEDArrayItf

Description

`XALEDArrayItf` is used to activate / deactivate the LEDs, as well as to set the color of LEDs, if supported.

`XALEDArrayItf` uses the following state model per LED, which indicates whether the LED is on or off:

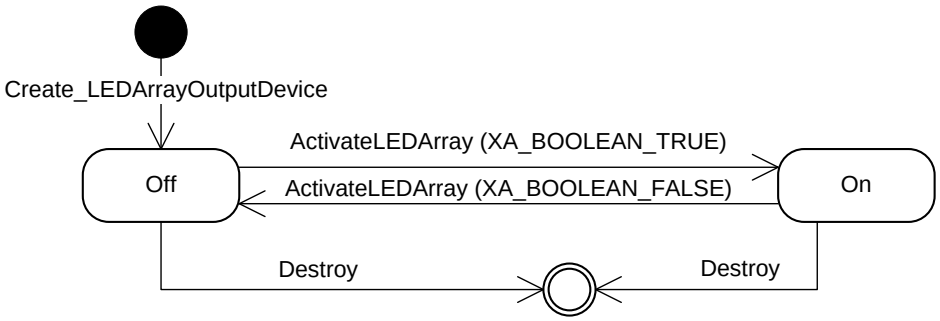


Figure 15: XALEDArrayItf state model

This interface is supported on the LED Array (see section 7.3) object.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_LED;

struct XALEDArrayItf_;
typedef const struct XALEDArrayItf_ * const * XALEDArrayItf;

struct XALEDArrayItf_ {
    XAresult (*ActivateLEDArray) (
        XALEDArrayItf self,
        XAuint32 lightMask
    );
    XAresult (*IsLEDArrayActivated) (
        XALEDArrayItf self,
        XAuint32 * pLightMask
    );
    XAresult (*SetColor) (
        XALEDArrayItf self,
        XAuint8 index,
        const XAHSL * pColor
    );
    XAresult (*GetColor) (
        XALEDArrayItf self,
        XAuint8 index,
        XAHSL * pColor
    );
};
```

Interface ID

a534d920-f775-11db-8b70-0002a5d5c51b

Defaults

Initially, all LEDs are in the off state. Default color is undefined.

Methods

ActivateLEDArray			

`XAresult (*ActivateLEDArray) (`
`  XALEDArrayItf self,`
`  XAuint32 lightMask`
`);`			

Description	Activates or deactivates individual LEDs in an array of LEDs.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`lightMask`	[in]	Bit mask indicating which LEDs should be activated or deactivated.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTROL_LOST`		

Comments	Valid bits in `lightMask` range from the least significant bit, which indicates the first LED in the array, to bit `XALEDDescriptor::ledCount` – 1, which indicates the last LED in the array. Bits set outside this range are ignored.		
See also	None.		

IsLEDArrayActivated

```
XAresult (*IsLEDArrayActivated) (
    XALEDArrayItf self,
    XAuint32 * pLightMask
);
```

Description	Returns the state of each LED in an array of LEDs.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pLightMask`	[out]	Address to store a bit mask indicating which LEDs are activated or deactivated.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Valid bits in `pLightMask` range from the least significant bit, which indicates the first LED in the array, to bit `XALEDDescriptor::ledCount` – 1, which indicates the last LED in the array. Bits set outside this range are ignored.		
See also	None.		

SetColor

```
XAresult (*SetColor) (
    XALEDArrayItf self,
    XAuint8 index,
    const XAHSL * pColor
);
```

Description	Sets the color of an individual LED.		
Pre-conditions	The LED must support setting color, per `XALEDDescriptor::colorMask`.		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Index of the LED. Range is [0, `XALEDDescriptor::ledCount`)
	`pColor`	[in]	Address of a data structure containing an HSL color.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTROL_LOST`		

Comments	None.		
See also	None.		

GetColor			
XAresult (*GetColor) (**XALEDArrayItf self,** **XAuint8 index,** **XAHSL * pColor** **);**			
Description	Returns the color of an individual LED.		
Pre-conditions	The LED must support setting color, per `XALEDDescriptor::colorMask`.		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Index of the LED. Range is [0, `XALEDDescriptor::ledCount`)
	`pColor`	[out]	Address to store a data structure containing an HSL color.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	None.		

8.21 XAMetadataExtractionItf

Description

The `XAMetadataExtractionItf` interface allows an application developer to acquire metadata. It is used for scanning through a file's metadata, providing the ability to determine how many metadata items are available, to filter for or against metadata items by key, and to have the engine fill in a data structure containing full metadata information for a metadata item.

The `XAMetadataExtractionItf` interface defaults to a simple search: in the case of simple formats (MP3, ADTS, WAVE, AU, AIFF, etc.), there is only one location for metadata, and this simple method searches it completely; in the case of advanced formats (MP4/3GP, XMF, SMIL, etc.), there are potentially many locations for metadata, and the engine searches only the topmost layer of metadata. Used in combination with the `XAMetadataTraversalItf` interface, the `XAMetadataExtractionItf` interface is able to search all metadata in any file using a variety of search modes.

This interface is mandated on Media Player objects (see section 7.4) and implicit on Metadata Extractor (see section 7.6) objects. See section D.6 for an example using this interface.

Dynamic Interface Addition

If this interface is added dynamically (using `DynamicInterfaceManagementItf`) the set of exposed metadata items might be limited compared to the set of exposed items had this interface been requested during object creation time. Typically, this might be the case in some implementations for efficiency reasons, or when the interface is added dynamically during playback of non-seakable streamed content and the metadata is located earlier in the stream than what was the interface addition time.

Khronos Keys

The keys that can be used to access metadata are the keys defined in the metadata specification of the media format in question. In addition, the OpenMAX AL specification defines a few format-agnostic keys, called "Khronos keys". The Khronos keys are for those developers who may not be familiar with the original metadata keys of the various media formats, but still want to extract metadata using OpenMAX AL and OpenMAX AL. It is the responsibility of API implementations to map these Khronos keys to the format-specific standard metadata keys. The Khronos keys are not meant to replace the standard metadata keys or to restrict the number of metadata keys available to the application. Developers conversant with the standard metadata keys in each format can still specify exactly the keys they are interested in with the help of the `MetadataExtractionItf`. The support for these Khronos keys is format-dependent.

The following table lists the Khronos keys. This list does not purport to be a comprehensive union of the standard keys in the various media formats. On the contrary, it is deliberately limited to the set of commonly-used metadata items. It should be considered as a baseline list.

Table 11: Khronos Keys

"KhronosTitle"	The title of the low-level entity, such as the name of the song, book chapter, image, video clip.
"KhronosAlbum"	The title of the high-level entity, such as the name of the song/video/image album, the name of the book.

“KhronosTrackNumber”	The number of the track.
“KhronosArtist”	The name of the artist, performer.
“KhronosGenre”	The genre of the media.
“KhronosYear”	The release year.
“KhronosComment”	Other comments on the media. For example, for images, this could be the event at which the photo was taken, etc.
“KhronosArtistURL”	A URL pointing to the artist’s site.
“KhronosContentURL”	A URL pointing to the site from which (alternate versions of) the content can be downloaded.
“KhronosRating”	A subjective rating of the media.
“KhronosAlbumArtJPEG”	Associated JPEG image, such as album art. The value associated with this key (the image itself) is in binary, in one of several image formats.
“KhronosAlbumArtPNG”	Associated PNG image, such as album art. The value associated with this key (the image itself) is in binary, in one of several image formats.
“KhronosCopyright”	Copyright text.
“KhronosSeekPoint”	Seek points of the media.

In this regard, three important scenarios are worth considering:

Scenario 1: Some of the Khronos keys do not have an equivalent standard metadata key in the media format under consideration: Only those Khronos keys for which there exists a mapping to the standard keys of the media are populated; the remaining Khronos keys remain empty, that is, no mapping exists and they are not exposed.

Scenario 2: The application is interested in metadata keys that are not part of the list of Khronos keys: The application has the option of ignoring the Khronos keys entirely and directly specifying exactly those standard metadata keys that it is interested in, using `XAMetadataExtractionItf`.

Scenario 3: The application’s metadata key list of interest is a proper superset of the Khronos key list: The application has the option of ignoring the Khronos key list entirely (as in Scenario #2) or it can use the Khronos key list and supplement it by accessing the extra format-specific standard keys directly using the `XAMetadataExtractionItf`.

All the Khronos keys are encoded in ASCII. The encoding and the language country code of the associated values depend on the media content. However, the encoding of the values is in one of the encoded strings with an exception that the values associated with “KhronosAlbumArtJPEG” and “KhronosAlbumArtPNG” keys have the encoding `XA_CHARACTERENCODING_BINARY`.

Seek Points

`XAMetadataExtractionItf` can be used for querying the seek points of the media. This is done by using the standard metadata (ASCII) key “KhronosSeekPoint”.

The associated value of Khronos seek points are represented with `XAMetadataInfo` structures, which is the case with all the metadata keys. The character encoding of this `XAMetadataInfo` structure is `XA_CHARACTERENCODING_BINARY`, since the value has special format described in Figure 16.

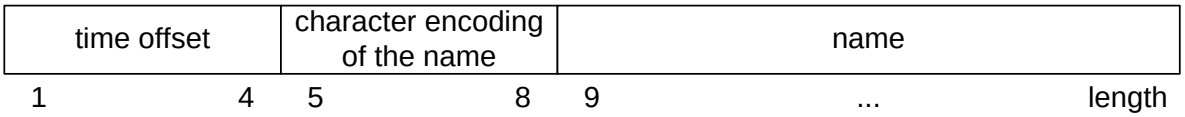


Figure 16: Fields of "KhronosSeekPoint" XAMetadataInfo

The data field of the `XAMetadataInfo` struct contains in its first 4 bytes the time offset (little endian) of the seek point as `XAmilliseconds`. (The length of the value is 4 bytes, since `XAmillisecond` is `XAuint32`.) `SeekItf::SetPosition()` can be used for seeking that seek point.

The bytes from the 5th to the 8th contain the character encoding of the name of the seek point as a `XA_CHARACTERENCODING` macro.

Starting from the 9th byte, the data field contains the name of the seek point (for example, the chapter name) in the character encoding defined in bytes 5 to 8 and the language defined in the `XAMetadataInfo` struct. The name is always null-terminated, which means that even if the name would be empty, the length of the value is always at least 9 bytes.

There can be multiple "KhronosSeekPoint" items for the same seek point to allow multiple language support. That is, the number of "KhronosSeekPoint" items is the number of seek points times the number of languages supported. The `AddKeyFilter()` method can be used for looking at seek points only in specific language by setting the pKey parameter as "KhronosSeekPoint" and the `valueLangCountry` parameter to contain the language / country code of interest.

Mandated Keys

An implementation of `XAMetadataExtractionItf` must support all methods on the interface. This specification does not mandate that an implementation support any particular key (Khronos key or otherwise) even in cases where the interface itself is mandated on an object.

Filtering of Metadata Items

The interface enables filtering of metadata items according to several criteria (see `AddKeyFilter()`). Theoretically, the application may never use the filtering functionality and do filtering itself. However, in practice, an implementation may use the filtering information in order to make extraction more efficient in terms of memory consumption or computational complexity. For that matter, it is recommended that applications that are not interested in the entire set of metadata items will use the filtering mechanism.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_METADATAEXTRACTION;

struct XAMetadataExtractionItf_;
typedef const struct XAMetadataExtractionItf_
    * const * XAMetadataExtractionItf;

struct XAMetadataExtractionItf_ {
    XAresult (*GetItemCount) (
        XAMetadataExtractionItf self,
        XAuint32 * pItemCount
    );
    XAresult (*GetKeySize) (
        XAMetadataExtractionItf self,
        XAuint32 index,
        XAuint32 * pKeySize
    );
    XAresult (*GetKey) (
        XAMetadataExtractionItf self,
        XAuint32 index,
        XAuint32 keySize,
        XAMetadataInfo * pKey
    );
    XAresult (*GetValueSize) (
        XAMetadataExtractionItf self,
        XAuint32 index,
        XAuint32 * pValueSize
    );
    XAresult (*GetValue) (
        XAMetadataExtractionItf self,
        XAuint32 index,
        XAuint32 valueSize,
        XAMetadataInfo * pValue
    );
    XAresult (*AddKeyFilter) (
        XAMetadataExtractionItf self,
        XAuint32 keySize,
        const void * pKey,
        XAuint32 keyEncoding,
        const XAchar * pValueLangCountry,
        XAuint32 valueEncoding,
        XAuint8 filterMask
    );
    XAresult (*ClearKeyFilter) (
        XAMetadataExtractionItf self
    );
};
```

Interface ID

5df4fda0-f776-11db-abc5-0002a5d5c51b

Defaults

The metadata key filter is empty upon realization of the interface. The default metadata scope is the root of the file.

Methods

GetItemCount			

`XAresult (*GetItemCount) (`
`  XAMetadataExtractionItf self,`
`  XAuint32 * pItemCount`
`);`			

Description	Returns the number of metadata items within the current scope of the object.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pItemCount`	[out]	Number of metadata items. Must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	`itemCount` is determined by the current metadata filter. For example, in a situation where four metadata items exist, and there is no filter, `GetItemCount()` will return 4; if there is a filter that matched only one of the keys, `GetItemCount()` will return 1. `GetItemCount()` returns the number of metadata items for a given metadata scope (active node). The scope is determined by methods within `XAMetadataTraversalItf`.		
See also	None		

GetKeySize			
XAresult (*GetKeySize) (**XAMetadataExtractionItf self,** **XAuint32 index,** **XAuint32 * pKeySize** **);**			
Description	Returns the byte size of a given metadata key.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Metadata item Index. Range is [0, `GetItemCount`).
	`pKeySize`	[out]	Address to store key size. `size` must be greater than 0. Must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	`GetKeySize()` is used for determining how large a block of memory is necessary to hold the key returned by `GetKey()`.		
See also	`GetKey()`		

GetKey			
XAresult (*GetKey) (**XAMetadataExtractionItf self,** **XAuint32 index,** **XAuint32 keySize,** **XAMetadataInfo * pKey** **);**			
Description	Returns a `XAMetadataInfo` structure and associated data referenced by the structure for a key.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Metadata item Index. Range is [0, `GetItemCount()`).
	`keySize`	[in]	Size of the memory block passed as `key`. Range is [1, `GetKeySize`].
	`pKey`	[out]	Address to store the key. Must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	`GetKey()` fills out the `XAMetadataInfo` structure, including data for the key beyond the size of the structure. If the given size is smaller than the needed size XA_RESULT_BUFFER_INSUFFICIENT is returned and only data of the given size will be written; however, no invalid strings are written. That is, the null-terminator always exists and multibyte characters are not cut in the middle.		
See also	`GetKeySize()`		

GetValueSize			
XAresult (*GetValueSize) (XAMetadataExtractionItf self, XAuint32 index, XAuint32 * pSize);			
Description	Returns the byte size of a given metadata value.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Metadata item Index. Range is [0, `GetItemCount()`).
	`pSize`	[out]	Address to store value size. `size` must be greater than 0. Must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	`GetValueSize()` is used for determining how large a block of memory is necessary to hold the value returned by `GetValue()`.		
See also	`GetValue()`		

GetValue			
XAresult (*GetValue) (**XAMetadataExtractionItf self,** **XAuint32 index,** **XAuint32 size,** **XAMetadataInfo * pValue** **);**			
Description	Returns a `XAMetadataInfo` structure and associated data referenced by the structure for a value.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Metadata item Index. Range is [0, `GetItemCount()`).
	`size`	[in]	Size of the memory block passed as `value`. Range is [0, `GetValueSize`].
	`pValue`	[out]	Address to store the `value`. Must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	`GetValue()` fills out the `XAMetadataInfo` structure, including data for the value beyond the size of the structure. If the given size is smaller than the needed size XA_RESULT_BUFFER_INSUFFICIENT is returned and only data of the given size will be written; however, no invalid strings are written. That is, the null-terminator always exists and multibyte characters are not cut in the middle.		
See also	`GetValueSize()`		

AddKeyFilter			
XAresult (*AddKeyFilter) (**XAMetadataExtractionItf self,** **XAuint32 keySize,** **const void * pKey,** **XAuint32 keyEncoding,** **const XAchar * pValueLangCountry,** **XAuint32 valueEncoding,** **XAuint8 filterMask** **);**			
Description	Adds a filter for a specific key.		
Pre-conditions	At least one criteria parameter (`key`, `keyEncoding`, `pValueLangCountry`, `valueEncoding`) must be provided.		
Parameters	`self`	[in]	Interface self-reference.
	`keySize`	[in]	Size, in bytes, of the `pKey` parameter. Ignored if filtering by key is disabled.
	`pKey`	[in]	The key to filter by. The entire key must match. Ignored if filtering by key is disabled.
	`keyEncoding`	[in]	Character encoding of the `pKey` parameter. Ignored if filtering by key is disabled.
	`pValueLangCountry`	[in]	Language / country code of the value to filter by. Ignored if filtering by language / country is disabled. See `XAMetadataInfo` structure in section 9.1.28.
	`valueEncoding`	[in]	Encoding of the value to filter by. Ignored if filtering by encoding is disabled.
	`filterMask`	[in]	Bitmask indicating which criteria to filter by. Should be one of the `XA_METADATA_FILTER` macros, see section 9.2.39.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

AddKeyFilter	
Comments	`AddKeyFilter()` adds a key to the metadata key filter. The filter defines which metadata items are available when asking how many exist (`GetItemCount()`) and how they are indexed for calls to `GetKeySize()`, `GetKey()`, `GetValueSize()`, and `GetValue()`. For example, if a file contains two metadata items, with keys "foo" and "bar" (both ASCII), calling `AddKeyFilter()` for "foo" will cause `GetItem` to return only the metadata item "foo". A subsequent call to `AddKeyFilter` for "bar" will cause `GetItem()` to return both metadata items. The key filter uses one or more of the following criteria: key data, value encoding, and language country specification. Key data filter will consider a metadata item to match when the data in the filter key charset encoding and filter key value fields are identical to the key charset encoding and key value, respectively, found in a metadata item in the media. If the filter key charset encoding is different from the charset encoding that the media metadata item uses, it is optional for the implementation to convert the values of the filter key and the media metadata item key to the same charset, and evaluate whether they match. Language / country filter will consider a metadata item to match the criteria if the item's value language / country matches the filter's language / country code. The value encoding filter will simply match all items with the same value encoding. While it is possible to use all three criteria when calling `AddKeyFilter()`, it is also possible to include fewer criteria. `filterMask` is used for defining which criteria should be considered when calling `AddKeyFilter()`. It is constructed by bit-wise ORing of the metadata filter macros, see section 9.2.39. Note that `AddKeyFilter()` treats parameters as if they were ANDed together. For example, calling `AddKeyFilter()` with key data and language / country code (but not encoding) means that the filter will cause metadata that matches both the key and the language / country code to be returned, but not metadata that matches only one. Further note that subsequent calls to `AddKeyFilter()` are treated as if they were Ored together. For example, if the first call passed a key (but nothing else) and a second call passed a key and an encoding (but no language / country code), the interface will return metadata matching the first key and metadata matching both the second key and the encoding. For example, to filter for all metadata that uses the ASCII encoding for the value, pass `valueEncoding` as `XA_CHARACTERENCODING_ASCII` and `filterMask` as `XA_METADATA_FILTER_ENCODING`. To filter for all metadata that uses the ASCII encoding for the value *and* uses the language country code "en-us", pass `valueEncoding` as `XA_CHARACTERENCODING_ASCII`, `valueLangCountry` as "en-us", and `filterMask` as `XA_METADATA_FILTER_ENCODING` | `XA_METADATA_FILTER_LANG`. Note that when the filter is clear (that is, when no filter criteria have been added or after they have been cleared), the filter is defined so that `GetItemCount()` returns all metadata items (as if each criteria was set to a wildcard).
See also	`ClearKeyFilter()`

ClearKeyFilter			
XAresult (*ClearKeyFilter) (XAMetadataExtractionItf self);			
Description	Clears the key filter.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	Note that when the filter is clear (that is, when no filter criteria have been added or after they have been cleared), the filter is defined so that `GetItemCount()` returns all metadata items (as if each criteria was set to a wildcard).		
See also	`AddKeyFilter()`		

8.22 XAMetadataInsertionItf

Description

This interface is for inserting/overwriting metadata to the media object. The actual inserting will happen once the output is written, (see `XASnapshotItf` and `XARecordItf` for details regarding exactly when the media is written to the sink). The metadata should have been set with this interface before the writing takes place.

Metadata Insertion for Tree-based Structures

For tree-based metadata structures (e.g. 3GPP or MP4 files), instead of using CreateChildNode to create the tree from scratch, the XAMetadataInsertionItf can be used in conjunction with the (optional) XAMetadataTraversalItf on the media recorder object to insert metadata items in the appropriate nodes in the tree. This requires that the implementation of the media recorder object detect the container format of the data sink that was specified at the time that the object was created (see the CreateMediaRecorder() method in Section 8.13), and create the appropriate metadata tree structure. The XAMetadataTraversalItf can then be used to traverse this tree structure, with traversal mode set to XA_METADATATRAVERSALMODE_NODE to determine the node IDs, and insert metadata items at the appropriate nodes using XAMetadataInsertionItf::InsertMetadataItem.

This mechanism has the following advantages: it frees the application developer from the burden of determining the correct metadata tree structure for a given container format, and ensures a higher-level of accuracy of the tree structure used (implementers typically have the wherewithal to use the correct structure).

XAMetadataTraversalItf is not needed for inserting metadata into flat metadata lists since XAMetadataInsertionItf can be used to insert all metadata items into the root node by just specifying the root note ID, XA_ROOT_NODE_ID. See Appendix D.5 for sample code on flat list metadata insertion.

This interface is a mandated interface of Media Recorder objects (see section 7.5). See section D.5 for an example using this interface.

Khronos Keys

In general, the keys that can be used to write metadata are the keys defined in the metadata specification of the media format in question. In addition, the OpenMAX AL specification defines a few format-agnostic keys, called “Khronos keys”. The Khronos keys are for developers who may not be familiar with the original metadata keys of the various media formats, but still want to insert metadata using OpenMAX AL. It is the responsibility of the API implementations to map these Khronos keys to the format-specific standard metadata keys. The Khronos keys are not meant to replace the standard metadata keys or to restrict the number of metadata keys available to the application. Developers conversant with the standard metadata keys in each format can still specify exactly the keys they are interested in with the help of the `MetadataInsertionItf`.
`MetadataInsertionItf::GetKeys()` can be used for querying which format-specific keys and Khronos keys are supported for writing by the implementation.

The support for the Khronos keys is format-dependent.

See `MetadataExtractionItf` (see section 8.21) for the definitions of the Khronos keys.

In this regard, three important scenarios are worth considering:

Scenario 1: Some of the Khronos keys do not have an equivalent standard metadata key in the media format under consideration: Just those Khronos keys for which there exists a mapping to the standard keys of the media can be available for writing and are exposed via method GetKeys.

Scenario 2: The application is interested in metadata keys that are not part of the list of Khronos keys: The application has the option of ignoring the Khronos keys entirely and directly specifying exactly those standard metadata keys that it is interested in, keeping in mind what `GetKeys()` method lists about their availability.

Scenario 3: The application's metadata key list of interest is a proper superset of the Khronos key list: The application has the option of ignoring the Khronos key list entirely (as in Scenario #2) or it can use the Khronos key list and supplement it by writing by the extra format-specific standard keys directly takin into account what keys `GetKeys()` method describes to be available for writing.

The encoding of the associated values to Khronos keys must be in one of the string encodings with an exception that the values associated with "KhronosAlbumArtJPEG" and "KhronosAlbumArtPNG" keys have to be in encoding `XA_CHARACTERENCODING_BINARY`. If this rule is not followed when Khronos keys are used for metadata insertion, the insertion for that node will fail and that will be indicated with `xaMetadataInsertionCallback()`.

Mandated Keys

An implementation of `XAMetadataInsertionItf` must merely support all methods on the interface. This specification does not mandate that an implementation supports writing any particular key (Khronos key or otherwise) even in cases where the interface itself is mandated on an object. The supported keys for writing can be queried with methods `GetSupportedKeysCount`, `GetKeySize` and `GetKeys()`.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_METADATAINSERTION;

struct XAMetadataInsertionItf_;
typedef const struct XAMetadataInsertionItf_
    * const * XAMetadataInsertionItf;

typedef struct XAMetadataInsertionItf_ {
    XAresult (*CreateChildNode) (
        XAMetadataInsertionItf self,
        XAint32 parentNodeID,
        XAuint32 type,
        XAchar * pMimeType,
        XAint32 * pChildNodeID
    );
    XAresult (*GetSupportedKeysCount) (
        XAMetadataInsertionItf self,
        XAint32 nodeID,
        XAboolean * pFreeKeys,
        XAuint32 * pKeyCount,
        XAuint32 * pEncodingCount
    );
    XAresult (*GetKeySize) (
        XAMetadataInsertionItf self,
        XAint32 nodeID,
        XAuint32 keyIndex,
        XAuint32 * pKeySize
    );
    XAresult (*GetKey) (
        XAMetadataInsertionItf self,
        XAint32 nodeID,
        XAuint32 keyIndex,
        XAuint32 keySize,
        XAMetadataInfo * pKey
    );
    XAresult (*GetFreeKeysEncoding) (
        XAMetadataInsertionItf self,
        XAint32 nodeID,
        XAuint32 encodingIndex,
        XAuint32 * pEncoding
    );
    XAresult (*InsertMetadataItem) (
        XAMetadataInsertionItf self,
        XAint32 nodeID,
        XAMetadataInfo * pKey,
        XAMetadataInfo * pValue,
        XAboolean overwrite
    );
    XAresult (*RegisterCallback) (
        XAMetadataInsertionItf self,
        xaMetadataInsertionCallback callback,
        void * pContext
    );
};
```

Interface ID

0fe726a0-d71b-11df-9d6c-0002a5d5c51b

Defaults

Not applicable.

Callback

xaMetadataInsertionCallback			
typedef void (XAAPIENTRY * xaMetadataInsertionCallback) (XAMetadataInsertionItf caller, void * pContext, XAMetadataInfo * pKey, XAMetadataInfo * pValue, XAint32 nodeID, XAboolean result);			
Description	Callback function called on completion of actual writing of a metadata key/value pair.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`pKey`	[in]	The key that was written (or that was tried to be written in case of failure). The application can now free the memory allocated for this struct.
	`pValue`	[in]	The value that was written (or that was tried to be written in case of failure). The application can now free the memory allocated for this struct.
	`nodeID`	[in]	ID of the node where this key/value pair is (or would be in case of failure).
	`result`	[in]	True, if writing of the metadata key/value pair was successful; false otherwise.
Comments	This is not a method of the interface but is the callback description and prototype.		
See Also	`RegisterCallback()`		

Methods

CreateChildNode			
XAresult (*CreateChildNode) (XAMetadataInsertionItf self, XAint32 parentNodeID, XAuint32 type, XAchar * pMimeType, XAint32 * pChildNodeID);			
Description	Creates a new child node for the given parent. This child can be utilized with `InsertMetadataItem()` method. This method does not actually write yet anything on media object. The actual writing and creation of the node will happen only if the node id is given to `InsertMetadataItem()` method as argument.		
Pre-conditions	none		
Parameters	`self`	[in]	Interface self-reference.
	`parentNodeID`	[in]	ID of the parent node for this new child node.
	`type`	[in]	Type of the new node. See XA_NODETYPE macros.
	`pMimeType`	[in]	Suggested MIME-type for this new child node. The given MIME-type is a hint only and the implementation might override it. The application is allowed to provide a zero-length string here if the application doesn't want to specify the MIME-type.
	`pChildNodeID`	[out]	ID of the created child node.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTENT_UNSUPPORTED`		

Comments	`XA_RESULT_FEATURE_UNSUPPORTED` is returned if the media object does not support tree-like metadata. That is, writing metadata only to the root is supported and therefore creating new nodes is not supported. `XA_RESULT_CONTENT_UNSUPPORTED` is returned if the media object does not support the given node type.		

GetSupportedKeysCount

```
XAresult (*GetSupportedKeysCount) (
    XAMetadataInsertionItf self,
    XAint32 nodeID,
    XAboolean * pFreeKeys,
    XAuint32 * pKeyCount,
    XAuint32 * pEncodingCount
);
```

Description

A query method to tell if the metadata keys (for writing metadata) can be freely chosen by the application or if they are fixed (for the given node).

If the implementation supports only fixed set of keys for metadata writing for the format and node in question, this query method gives the number of the supported fixed keys and `GetKeySize` and `GetKey` can then be used to query those keys.

On the other hand, if the implementation supports free keys for metadata writing for the format and node in question, this query method gives the number of supported character encodings and `GetFreeKeysEncoding` can then be used to query those encodings.

Pre-conditions

None

Parameters

`self`	[in]	Interface self-reference.
`nodeID`	[in]	ID of the node whose supported keys are queried. XA_ROOT_NODE_ID is used to refer to the root node of the container.
`pFreeKeys`	[out]	True if keys can be freely chosen by the application; false if the keys are fixed.
`pKeyCount`	[out]	If `pFreeKeys` is false, this is the number of keys available for writing; if `pFreeKeys` is true, this is the number of commonly used keys for this node. Please use GetKeySize and GetKey methods to query the keys one at a time.
`pEncodingCount`	[out]	If `pFreeKeys` is false, this value should be ignored; if `pFreeKeys` is true, this is the number of supported character encodings for this node. Please use `GetFreeKeysEncoding` method to query them one at a time.

Return value

The return value can be one of the following:

`XA_RESULT_SUCCESS`

`XA_RESULT_PARAMETER_INVALID`

Comments

none

GetKeySize			
XAresult (*GetKeySize) (**XAMetadataInsertionItf self,** **XAint32 nodeID,** **XAuint32 index,** **XAuint32 * pKeySize** **);**			
Description	Returns the byte size required for a supported metadata key pointed by the given index.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`nodeID`	[in]	ID of the node whose supported keys are queried.
	`index`	[in]	Index for supported metadata keys. Range is [0, KeyCount-1].
	`pKeySize`	[out]	Address to store key size. size must be greater than 0. Must be non-NULL.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	`GetKeySize()` is used for determining how large a block of memory is necessary to hold the key returned by `GetKey()`.		
See also	`GetKey()`		

<table>
<tr><th colspan="4">GetKey</th></tr>
<tr><td colspan="4"><pre>
XAresult (*GetKey) (
    XAMetadataInsertionItf self,
    XAint32 nodeID,
    XAuint32 index,
    XAuint32 keySize,
    XAMetadataInfo * pKey
);
</pre></td></tr>
<tr><td>Description</td><td colspan="3">Returns a XAMetadataInfo structure and associated data referenced by the structure for a supported key.</td></tr>
<tr><td>Pre-conditions</td><td colspan="3">None</td></tr>
<tr><td rowspan="5">Parameters</td><td>self</td><td>[in]</td><td>Interface self-reference.</td></tr>
<tr><td>nodeID</td><td>[in]</td><td>ID of the node whose supported keys are queried.</td></tr>
<tr><td>index</td><td>[in]</td><td>Index for supported metadata keys. Range is [0, KeyCount-1].</td></tr>
<tr><td>keySize</td><td>[in]</td><td>Size of the memory block passed as key. Range is [1, GetKeySize].</td></tr>
<tr><td>pKey</td><td>[out]</td><td>Address to store the key. Must be non-NULL.</td></tr>
<tr><td>Return value</td><td colspan="3">The return value can be one of the following:
XA_RESULT_SUCCESS
XA_RESULT_PARAMETER_INVALID
XA_RESULT_BUFFER_INSUFFICIENT</td></tr>
<tr><td>Comments</td><td colspan="3">GetKey() fills out the XAMetadataInfo structure, including data for the key beyond the size of the structure.
If the given size is smaller than the needed size XA_RESULT_BUFFER_INSUFFICIENT is returned and only data of the given size will be written; however, no invalid strings are written. That is, the null-terminator always exists and multibyte characters are not cut in the middle.</td></tr>
<tr><td>See also</td><td colspan="3">GetSupportedKeysCout(), GetKeySize()</td></tr>
</table>

GetFreeKeysEncoding			
XAresult (*GetFreeKeysEncoding) (**XAMetadataInsertionItf self,** **XAint32 nodeID,** **XAuint32 * pEncodingIndex,** **XAuint32 * pEncoding,** **);**			
Description	A method to be used in case implementation supports free keys for metadata insertion. This method tells supported character encodings for those free keys.		
Pre-conditions	`pFreeKeys` (in `GetSupportedKeysCount`) should be true.		
Parameters	`self`	[in]	Interface self-reference.
	`nodeID`	[in]	ID of the node whose supported keys are queried.
	`pEncodingIndex`	[in]	Metadata insertion free keys encodings Index. Range is [0, EncodingCount-1].
	`pEncoding`	[out]	A supported character encoding for free keys to be written. See XA_CHARACTERENCODING macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	none		
See also	`GetSupportedKeysCount()`		

InsertMetadataItem

```
XAresult (*InsertMetadataItem) (
    XAMetadataInsertionItf self,
    XAint32 nodeID,
    XAMetadataInfo * pKey,
    XAMetadataInfo * pValue,
    XAboolean overwrite
);
```

Description	Inserts the key/value pair to the specified node of the metadata tree. (If the specified node does not exist in media object, the key/value pair cannot be written.) Giving `XA_ROOT_NODE_ID` as the node, writes the key/value pair to the root. overwrite flag tells what to do if the there is already a value set for the given key in the given node. (For example, camera device might write automatically some metadata to the resulting image.) Please note that in some formats, the `langCountry` field of `XAMetadataInfo` structs will be ignored.		
Pre-conditions	none		
Parameters	`self`	[in]	Interface self-reference.
	`nodeID`	[in]	ID of the node whereto the metadata is to be written. ID given by `XAMetadataTraversalItf::GetChildInfo()` can be used here, if `XAMetadataTraversalItf` is available and the node exists already. Otherwise, `CreateChildNode()` should be used to get the node ID.
	`pKey`	[in]	Key to be written. The application should not deallocate or change the content of this struct before receiving the corresponding `xaMetadataInsertionCallback` or destroying the object.
	`pValue`	[in]	Value to be written. The application should not deallocate or change the content of this struct before receiving the corresponding `xaMetadataInsertionCallback` or destroying the object.
	`overwrite`	[in]	This flag is used to coordinate the insertion of the same metadata information being provided by both the source and application. If true, the information provided by the application shall be used. If false, the information provided by the source shall be used. Note: This parameter is only applicable when the same metadata information will be available from both the source and application.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	none		

RegisterCallback			
XAresult (*RegisterCallback) (struct XAMetadataInsertionItf _* self, xaMetadataInsertionCallback callback, void * pContext);			
Description	Registers a callback on the object that executes after each of the actual writings of metadata key/value pairs takes place.		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the result callback. If `NULL` then the callback is disabled.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	none		
See also	`xaMetadataInsertionCallback()`		

8.23 XAMetadataMessageItf

Description

The metadata message interface is used to set metadata callbacks. The user will be notified via callback of any metadata that is encountered during playback.

This interface on its own has no function or utility. It is designed for use in conjunction with `XAMetadataExtractionItf`, and is only able to be realized or dynamically added to an object when `XAMetadataExtractionItf` is already present.

The configuration state of the corresponding `XAMetadataExtractionItf` directly effects the performance of `XAMetadataMessageItf`. For example, any filters that are active will cause metadata during playback to be filtered prior to its exposure via callback.

This interface is supported on the Media Player [see section 7.4] and Metadata Extractor [see section 7.6].

Prototype

```
XA_API extern const XAInterfaceID XA_IID_METADATAMESSAGE;

struct XAMetadataMessageItf_;
typedef const struct XAMetadataMessageItf_ * const * XAMetadataMessageItf;

struct XAMetadataMessageItf_ {
   XAresult (*RegisterMetadataCallback) (
         XAMetadataMessageItf self,
         xaMetadataCallback callback,
         void *pContext
   );
};
```

Interface ID

7a8ff2f0-2033-11e0-ac64-0800200c9a66

Callbacks

xaMetadataCallback	

`typedef void (XAAPIENTRY *xaMetadataCallback){`
`XAMetadataMessageItf caller,`
`void *pContext,`
`XAuint32 index`
`);`	

Description	Executes when a Metadata element is encountered during playback.

xaMetadataCallback			
Parameters	caller	[in]	Interface instantiation on which the callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`index`	[in]	Metadata item Index for the metadata encountered. Interaction with this Index is done via the `XAMetadataExtractionItf` interface that must accompany the `XAMetadataMessageItf` interface. Additionally, this Index is only guaranteed to be valid and fixed for the duration of the callback.

Methods

RegisterMetadataCallback			
XAresult (*RegisterMetadataCallback) (**XAMetadataMessageItf self,** **xaMetadataCallback callback,** **void *pContext** **);**			
Description	Sets or clears the player's metadata callback.		
Parameters	self	[in]	Interface self-reference
	`callback`	[in]	Address of the metadata callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

See also	`xaMetadataCallback`		

8.24 XAMetadataTraversalItf

Description

The `XAMetadataTraversalItf` interface is used in order to support advanced metadata extraction. It allows developers to traverse a file using a variety of modes, which determine how to traverse the metadata and define how the methods within the interface behave.

The interface provides the ability to set the traversal mode, to determine how many child nodes exist within a given scope and what their type is, and to set the scope.

This interface is a mandated interface of Media Player objects (see section 7.4) and implicit on Metadata Extractor (see section 7.6) objects.

Dynamic Interface Addition

If this interface is added dynamically (using `XADynamicInterfaceManagementItf`) the set of exposed metadata nodes might be limited compared to the set of exposed nodes had this interface been requested during object creation time. Typically, this might be the case in some implementations for efficiency reasons, or when the interface is added dynamically during playback of non-seakable streamed content and the metadata is located earlier in the stream than what was the interface addition time.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_METADATATRAVERSAL;

struct XAMetadataTraversalItf_;
typedef const struct XAMetadataTraversalItf_
 *  const *  XAMetadataTraversalItf;

struct XAMetadataTraversalItf_ {
    XAresult (*SetMode) (
        XAMetadataTraversalItf self,
        XAuint32 mode
    );
    XAresult (*GetChildCount) (
        XAMetadataTraversalItf self,
        XAuint32 * pCount
    );
    XAresult (*GetChildMIMETypeSize) (
        XAMetadataTraversalItf self,
        XAuint32 index,
        XAuint32 * pSize
    );
    XAresult (*GetChildInfo) (
        XAMetadataTraversalItf self,
        XAuint32 index,
        XAint32 * pNodeID,
        XAuint32 * pType,
        XAuint32 size,
        XAchar * pMimeType
    );
```

```
    XAresult (*SetActiveNode) (
        XAMetadataTraversalItf self,
        XAuint32 index
    );
};
```

Interface ID

73ffb0e0-f776-11db-a00e-0002a5d5c51b

Defaults

The metadata traversal mode defaults to XA_METADATATRAVERSALMODE_NODE. The default metadata scope is the root of the file. The active node is root.

Methods

SetMode			
XAresult (*SetMode) (**XAMetadataTraversalItf self,** **XAuint32 mode** **);**			
Description	Sets the metadata traversal mode.		
Pre-conditions	None		
Parameters	self	[in]	Interface self-reference.
	mode	[in]	Mode of metadata traversal.
Return value	The return value can be one of the following: XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID		
Comments	Metadata traversal mode determines how a file is parsed for metadata. It is possible to traverse the file either by iterating through the file in tree fashion - by node (XA_METADATATRAVERSALMODE_NODE, the default mode), or by scanning through the file as if it were a flat list of metadata items (XA_METADATATRAVERSALMODE_ALL). The optimal mode is largely determined by the file format.		
See also	XA_METADATATRAVERSALMODE		

GetChildCount

```
XAresult (*GetChildCount) (
    XAMetadataTraversalItf self,
    XAuint32 * pCount
);
```

Description	Returns the number of children (nodes, streams, etc.) within the current scope.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pCount`	[out]	Number of children.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Child count is determined by the metadata traversal mode: If the mode is set to `XA_METADATATRAVERSALMODE_ALL`, GetChildCount() will always return 0. If the mode is set to `XA_METADATATRAVERSALMODE_NODE`, GetChildCount() will return the number of nodes within the current scope. For example, in a Mobile XMF file with one SMF node and one Mobile DLS node, GetChildCount() will return 2 from the root.		
See also	`SetMode()`, `XA_METADATATRAVERSALMODE`		

GetChildMIMETypeSize

```
XAresult (*GetChildMIMETypeSize) (
    XAMetadataTraversalItf self,
    XAuint32 index,
    XAuint32 * pSize
);
```

Description	Returns the size in bytes needed to store the MIME type of a child.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Child index. Range is [0, `GetChildCount()`).
	`pSize`	[out]	Size of the MIME type in bytes.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`GetChildCount()`		

GetChildInfo			
XAresult (*GetChildInfo) (**XAMetadataTraversalItf self,** **XAuint32 index,** **XAint32 * pNodeID,** **XAuint32 * pType,** **XAuint32 size,** **XAchar * pMimeType** **);**			
Description	Returns information about a child.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Child index. Range is [0, `GetChildCount()`).
	`pNodeID`	[out]	Unique identification number of the child.
	`pType`	[out]	Node type. See XA_NODETYPE macro.
	`size`	[in]	Size of the memory block passed as `mimeType`. Range is (0, max `GetChildMIMETypeSize()`].
	`pMimeType`	[out]	Address to store the MIME type.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	To ignore MIME type, set `size` to 0 and `mimeType` to `NULL`.		
See also	`GetChildCount()`		

SetActiveNode			

`XAresult (*SetActiveNode) (`
`  XAMetadataTraversalItf self,`
`  XAuint32 index`
`);`			

Description	Sets the scope to a child index.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Child index. Range is special (see below).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	`SetActiveNode()` causes the current scope to descend or ascend to the given index. To descend, set `index` to [0, `GetChildCount()`). To ascend to the parent scope, set `index` to `XA_NODE_PARENT`. Calling `SetActiveNode()` with `index` set to `XA_NODE_PARENT` will return `XA_RESULT_PARAMETER_INVALID` if the active node is root.		
See also	`GetChildCount()`, `XA_NODE_PARENT`		

8.25 XAObjectItf

Description

The `XAObjectItf` interface provides essential utility methods for all objects. Such functionality includes the destruction of the object, realization and recovery, acquisition of interface pointers, a callback for runtime errors, and asynchronous operation termination.

A maximum of one asynchronous operation may be performed by an object at any given time. Trying to invoke an asynchronous operation when an object is already processing an asynchronous call is equivalent to aborting the first operation, then invoking the second one.

`XAObjectItf` is an implicit interface of all object types and is automatically available upon creation of every object.

Please refer to section 3.1.1 for details on the object states.

This interface is supported on all objects (see section 7).

Priority

This interface exposes a control for setting an object's priority relative to the other objects under control of the same instance of the engine. This priority provides a hint that the implementation can use when there is resource contention between objects.

Given resource contention between objects, an implementation will give preference to the object with the highest priority. This may imply that the implementation takes resources from one object to give to another if the two objects are competing for the same resources and the latter has higher priority. Given objects of identical priority competing for resources, the implementation steals the resources from the object that acquired them earlier to give to the object that requested them most recently.

Different objects may require entirely different resources. For this reason, it is possible that an object of high priority may have its resources stolen before an object of low priority. For example, a high-priority object may require access to dedicated hardware on the device while the low-priority object does not. If this dedicated hardware is demanded by the system, the resources may need to be stolen from the higher priority object, leaving the low priority object in the Realized state.

Loss of Control

Loss of control of an interface and its associated underlying resource means that the application has lost the ability to set the parameters of the interface and resource, but the interface and resource are otherwise still functioning and available for the application to use.

For example, imagine a system with a maximum of one environmental reverb unit (limited to one due to memory/CPU constraints, or because it is implemented in a hardware DSP). An application configures the environmental reverb for its output mix A. Another higher-priority application configures the environmental reverb for its own output mix B. The implementation may now signal to the first application that output mix A has lost control of the reverb (it is still applied on A, but with the same parameters as on B).

See the related object event macros (`XA_OBJECT_EVENT_ITF_CONTROL_TAKEN`, `XA_OBJECT_EVENT_ITF_CONTROL_RETURNED` and `XA_OBJECT_EVENT_ITF_PARAMETERS_CHANGED`) and the error code `XA_RESULT_CONTROL_LOST` for details.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_OBJECT;

struct XAObjectItf_;
typedef const struct XAObjectItf_ * const * XAObjectItf;

struct XAObjectItf_ {
    XAresult (*Realize) (
        XAObjectItf self,
        XAboolean async
    );
    XAresult (*Resume) (
        XAObjectItf self,
        XAboolean async
    );
    XAresult (*GetState) (
        XAObjectItf self,
        XAuint32 * pState
    );
    XAresult (*GetInterface) (
        XAObjectItf self,
        const XAInterfaceID iid,
        void * pInterface
    );
    XAresult (*RegisterCallback) (
        XAObjectItf self,
        xaObjectCallback callback,
        void * pContext
    );
    void (*AbortAsyncOperation) (
        XAObjectItf self
    );
    void (*Destroy) (
        XAObjectItf self
    );
    XAresult (*SetPriority) (
        XAObjectItf self,
        XAuint32 priority
    );
    XAresult (*GetPriority) (
        XAObjectItf self,
        XAuint32 * pPriority
    );
    XAresult (*SetLossOfControlInterfaces) (
        XAObjectItf self,
        XAuint16 numInterfaces,
        const XAInterfaceID * pInterfaceIDs,
        XAboolean enabled
    );
};
```

Interface ID

`99a1ce50-cb4e-11df-bd3b-0800200c9a66`

Defaults

The object is in Unrealized state.

No callback is registered.

Priority: `XA_PRIORITY_NORMAL`

Preemptable by object of same priority that is realized later than this object: `XA_BOOLEAN_FALSE`

Callbacks

xaObjectCallback			
typedef void (XAAPIENTRY * xaObjectCallback) (XAObjectItf caller, const void * pContext, XAuint32 event, XAresult result, XAuint32 param, void * pInterface);			
Description	A callback function, notifying of a runtime error, termination of an asynchronous call or change in the object's resource state.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`event`	[in]	One of the `XA_OBJECT_EVENT` macros.
	`result`	[in]	If `event` is `XA_OBJECT_EVENT_RUNTIME_ERROR`, `result` contains the error code. If `event` is `XA_OBJECT_EVENT_ASYNC_TERMINATION`, `result` contains the asynchronous function return code. For other values of `event`, this parameter should be ignored.
	`param`	[in]	If `event` is `XA_OBJECT_EVENT_RUNTIME_ERROR`, `XA_OBJECT_EVENT_RESOURCES_LOST` or `XA_OBJECT_EVENT_ASYNC_TERMINATION`, `param` contains the state of the object after the event. For other values of `event`, this parameter should be ignored.
	`pInterface`	[in]	If `event` is `XA_OBJECT_EVENT_ITF_CONTROL_TAKEN`, `XA_OBJECT_EVENT_ITF_CONTROL_RETURNED` or `XA_OBJECT_EVENT_ITF_PARAMETERS_CHANGED`, `pInterface` contains the interface affected. For other values of `event`, this parameter should be ignored.
Comments	Please note the restrictions applying to operations performed from within callback context, in section 3.3.		
See also	`RegisterCallback()`		

Methods

Realize			
XAresult (*Realize) (**XAObjectItf self,** **XAboolean async** **);**			
Description	Transitions the object from Unrealized state to Realized state, either synchronously or asynchronously.		
Pre-conditions	The object must be in Unrealized state.		
Parameters	`self`	[in]	Interface self-reference.
	`async`	[in]	If `XA_BOOLEAN_FALSE`, the method will block until termination. Otherwise, the method will return `XA_RESULT_SUCCESS`, and will be executed asynchronously. On termination, the `xaObjectCallback()` will be invoked, if registered, with the `XA_OBJECT_EVENT_ASYNC_TERMINATION`. The `result` parameter of the `xaObjectCallback()` will contain the result code of the function. However, if the implementation is unable to initiate the asynchronous call `XA_RESULT_RESOURCE_ERROR` will be returned.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_RESOURCE_ERROR`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_MEMORY_FAILURE`
	`XA_RESULT_IO_ERROR`
	`XA_RESULT_CONTENT_CORRUPTED`
	`XA_RESULT_CONTENT_UNSUPPORTED`
	`XA_RESULT_CONTENT_NOT_FOUND`
	`XA_RESULT_PERMISSION_DENIED`		

Comments	Realizing the object acquires the resources required for its functionality. The operation may fail if insufficient resources are available. In such a case, the application may wait until resources become available and a `XA_OBJECT_EVENT_RESOURCES_AVAILABLE` event is received, and then retry the realization. Another option is to try and increase the object's priority, thus increasing the likelihood that the object will steal another object's resources.		
See also	`None.`		

<table>
<tr><th colspan="4">Resume</th></tr>
<tr><td colspan="4">XAresult (*Resume) (
    XAObjectItf self,
    XAboolean async
);</td></tr>
<tr><td>Description</td><td colspan="3">Transitions the object from Suspended state to Realized state, either synchronously or asynchronously.</td></tr>
<tr><td>Pre-conditions</td><td colspan="3">The object must be in Suspended state.</td></tr>
<tr><td rowspan="2">Parameters</td><td>self</td><td>[in]</td><td>Interface self-reference.</td></tr>
<tr><td>async</td><td>[in]</td><td>If XA_BOOLEAN_FALSE, the method will block until termination. Otherwise, the method will return XA_RESULT_SUCCESS and will be executed asynchronously. On termination, the xaObjectCallback() will be invoked, if registered, with the XA_OBJECT_EVENT_ASYNC_TERMINATION. The result parameter of the xaObjectCallback() will contain the result code of the function. However, if the implementation is unable to initiate the asynchronous call XA_RESULT_RESOURCE_ERROR will be returned.</td></tr>
<tr><td>Return value</td><td colspan="3">The return value can be one of the following:
XA_RESULT_SUCCESS
XA_RESULT_RESOURCE_ERROR
XA_RESULT_PRECONDITIONS_VIOLATED</td></tr>
<tr><td>Comments</td><td colspan="3">Resuming the object acquires the resources required for its functionality. The operation may fail if insufficient resources are available. In such a case, the application may wait until resources become available and a XA_OBJECT_EVENT_RESOURCES_AVAILABLE event is received, and then retry the resumption. Another option is to try and increase the object's priority, thus increasing the likelihood that the object will steal another object's resources.</td></tr>
<tr><td>See also</td><td colspan="3">None.</td></tr>
</table>

GetState			
XAresult (*GetState) (**XAObjectItf self,** **XAuint32 * pState** **);**			
Description	Retrieves the current object state.		
Preconditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pState`	[out]	Pointer to the current state of the object. One of the object state macros, `XA_OBJECT_STATE`, will be written as result. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	None.		

GetInterface			
XAresult (*GetInterface) (**XAObjectItf self,** **const XAInterfaceID iid,** **void * pInterface** **);**			
Description	Obtains an interface exposed by the object		
Preconditions	The object is not in the Unrealized state.		
Parameters	`self`	[in]	Interface self-reference.
	`iid`	[in]	The interface type ID.
	`pInterface`	[out]	This should be a non-`NULL` pointer to a variable of the interface's type – for example, if a `XAObjectItf` is retrieved, this parameter should be of type `XAObjectItf *` type.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	If the object does not expose the requested interface type, the return code will be `XA_RESULT_FEATURE_UNSUPPORTED.`		
See also	None.		

RegisterCallback

```
XAresult (*RegisterCallback) (
    XAObjectItf self,
    xaObjectCallback callback,
    void * pContext
);
```

Description	Registers a callback on the object that executes when a runtime error occurs or an asynchronous operation terminates.		
Preconditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the result callback. If `NULL`, the callback is disabled.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The callback will report only runtime errors and results of calls to asynchronous functions.		
See also	`xaObjectCallback()`		

AbortAsyncOperation

```
void (*AbortAsyncOperation) (
    XAObjectItf self
);
```

Description	Aborts asynchronous call currently processed by the object. This method affects asynchronous calls initiated from `XAObjectItf` or `XADynamicInterfaceManagementItf` only. If no such call is being processed, the call is ignored. If a callback is registered, it will be invoked, with a `XA_OBJECT_EVENT_ASYNC_TERMINATION` as event and `XA_RESULT_OPERATION_ABORTED` as return code.		
Preconditions	None.		
Parameters	`self`	[in]	Interface self-reference.
Comments	The method is meant for graceful timeout or user-initiated abortion of asynchronous calls.		
See also	None.		

Destroy			
void (*Destroy) (XAObjectItf self);			
Description	Destroys the object.		
Preconditions	None.		
Parameters	`self`	[in]	Interface self-reference.
Comments	Destroy implicitly transfers the object through Unrealized state, thus freeing any resources allocated to the object prior to freeing it. All references to interfaces belonging to this object become invalid and may cause undefined behavior if used. All pending asynchronous operations are aborted, as if `AbortAsyncOperations()` has been called.		
See also	None.		

SetPriority			
XAresult (*SetPriority) (XAObjectItf self, XAuint32 priority);			
Description	Set the object's priority.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`priority`	[in]	The priority. The valid range for this parameter is [`INT_MIN`, `INT_MAX`]. The smaller the number, the higher the priority. Zero is the highest priority.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Although it is possible to set any priority within the specified range, `XA_PRIORITY` (see section 9.2.57) defines a fixed set of priorities for use with this method.		
See also	None.		

GetPriority			
XAresult (*GetPriority) (**struct XAObjectItf self,** **XAint32 * pPriority** **);**			
Description	Gets the object's priority.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pPriority`	[out]	Pointer to a location to receive the object's priority. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	None.		

SetLossOfControlInterfaces			
XAresult (*SetLossOfControlInterfaces) (**XAObjectItf self,** **XAint16 numInterfaces,** **const XAInterfaceID * pInterfaceIDs,** **XAboolean enabled** **);**			
Description	Sets/unsets loss of control functionality for a list of interface IDs. The default value of the enabled flag is determined by the global setting (see `XA_ENGINEOPTION_LOSSOFCONTROL` 9.2.32).		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`numInterfaces`	[in]	The length of the pInterfaceIDs array (ignored if pInterfaceIDs is `NULL`).
	`pInterfaceIDs`	[in]	Array of interface IDs representing the interfaces impacted by the enabled flag.
	`enabled`	[in]	If `XA_BOOLEAN_TRUE`, loss of control functionality is enabled for all interfaces represented by `pInterfaceIDs`. If `XA_BOOLEAN_FALSE`, loss of control functionality is disabled for all interfaces represented by `pInterfaceIDs`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	A call to this method overrides the global setting for loss of control functionality for the specified list of interfaces.		

8.26 XAOutputMixItf

Description

`XAOutputMixItf` is an interface for interacting with an output mix, including querying for the associated destination output devices, registering for the notification of changes to those outputs, and requesting changes to an output mix's associated devices.

This interface is supported on the Output Mix (see section 7.7) object.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_OUTPUTMIX;

struct XAOutputMixItf_;
typedef const struct XAOutputMixItf_ * const * XAOutputMixItf;

struct XAOutputMixItf_ {
    XAresult (*GetDestinationOutputDeviceIDs) (
        XAOutputMixItf self,
        XAint32 * pNumDevices,
        XAuint32 * pDeviceIDs
    );
    XAresult (*RegisterDeviceChangeCallback) (
        XAOutputMixItf self,
        xaMixDeviceChangeCallback callback,
        void * pContext
    );
    XAresult (*ReRoute) (
        XAOutputMixItf self,
        XAint32 numOutputDevices,
        const XAuint32 * pOutputDeviceIDs
    );
};
```

Interface ID

ab3c9c00-de20-11df-b8ef-0002a5d5c51b

Defaults

An output mix defaults to device ID values specific to the implementation.

Callbacks

xaMixDeviceChangeCallback			
typedef void (XAAPIENTRY * xaMixDeviceChangeCallback) (XAOutputMixItf caller, void * pContext);			
Description	Executes whenever an output mix changes its set of destination output devices. Upon this notification, the application may query for the new set of devices via the `XAOutputMixItf` interface.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
Comments	none		
See Also	`RegisterDeviceChangeCallback()`		

Methods

GetDestinationOutputDeviceIDs			
XAresult (*GetDestinationOutputDeviceIDs) (**XAOutputMixItf self,** **XAint32 * pNumDevices,** **XAuint32 * pDeviceIDs** **);**			
Description	Retrieves the device IDs of the destination output devices currently associated with the output mix.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumDevices`	[in/out]	As an input, specifies the length of the pDeviceIDs array (ignored if `pDeviceIDs` is `NULL`). As an output, specifies the number of destination output device IDs associated with the output mix.
	`pDeviceIDs`	[out]	Populated by the call with the list of deviceIDs (provided that `pNumDevices` is equal to or greater than the number of actual device IDs). If `pNumDevices` is less than the number of actual device IDs, the error code `XA_RESULT_BUFFER_INSUFFICIENT` is returned. Note: IDs may include `XA_DEFAULTDEVICEID_AUDIOOUTPUT`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	None		
See also	None		

RegisterDeviceChangeCallback

```
XAresult (*RegisterDeviceChangeCallback) (
    XAOutputMixItf self,
    xaMixDeviceChangeCallback callback
    void *  pContext,
);
```

Description	Registers a callback to notify the application when there are changes to the device IDs associated with the output mix.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Callback to receive the changes in device IDs associated with the output mix.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

ReRoute

```
XAresult (*ReRoute) (
  XAOutputMixItf self,
  XAint32 numOutputDevices,
  XAuint32 * pOutputDeviceIDs
);
```

Description	Requests a change to the specified set of output devices on an output mix.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`numOutputDevices`	[in]	Number of output devices specified.
	`pOutputDeviceIDs`	[in]	List of the devices specified. (Note: IDs may include `XA_DEFAULTDEVICEID_AUDIOOUTPUT`)

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	This method simply requests for a change in routing. The implementation may choose not to fulfill the request. If it does not fulfill the request, the method returns `XA_RESULT_FEATURE_UNSUPPORTED`.		

8.27 XAPlayItf

Description

`PlayItf` is an interface for controlling the playback state of an object. The playback state machine is as follows:

Table 12: Play Head Position in Different Play States

Play State	Head forced to beginning	Prefetching	Head trying to move
Stopped	X		
Paused		X	
Playing		X	X

This interface an implicit interface of Media Player objects (see section 7.4). See section D.2 for an example using this interface.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_PLAY;

struct XAPlayItf_;
typedef const struct XAPlayItf_ * const * XAPlayItf;

struct XAPlayItf_ {
    XAresult (*SetPlayState) (
        XAPlayItf self,
        XAuint32 state
    );
    XAresult (*GetPlayState) (
        XAPlayItf self,
        XAuint32 * pState
    );
    XAresult (*GetDuration) (
        XAPlayItf self,
        XAmillisecond * pMsec
    );
    XAresult (*GetPosition) (
        XAPlayItf self,
        XAmillisecond * pMsec
    );
    XAresult (*RegisterCallback) (
        XAPlayItf self,
        xaPlayCallback callback,
        void * pContext
    );
    XAresult (*SetCallbackEventsMask) (
        XAPlayItf self,
        XAuint32 eventFlags
    );
```

```
    XAresult (*GetCallbackEventsMask) (
        XAPlayItf self,
        XAuint32 * pEventFlags
    );
    XAresult (*SetMarkerPosition) (
        XAPlayItf self,
        XAmillisecond mSec
    );
    XAresult (*ClearMarkerPosition) (
        XAPlayItf self
    );
    XAresult (*GetMarkerPosition) (
        XAPlayItf self,
        XAmillisecond * pMsec
    );
    XAresult (*SetPositionUpdatePeriod) (
        XAPlayItf self,
        XAmillisecond mSec
    );
    XAresult (*GetPositionUpdatePeriod) (
        XAPlayItf self,
        XAmillisecond * pMsec
    );
};
```

Interface ID

b9c293e0-f776-11db-80df-0002a5d5c51b

Defaults

Initially, the playback state is XA_PLAYSTATE_STOPPED, the position is at the beginning of the content, the update period is one second, and there are no markers set nor callbacks registered and the callback event flags are cleared.

Callbacks

xaPlayCallback			
typedef void (XAAPIENTRY * xaPlayCallback) (XAPlayItf caller, void * pContext, XAuint32 event);			
Description	Notifies the player application of a playback event.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`event`	[in]	Indicates which event has occurred (see `XA_PLAYEVENT` macros).
Comments	None		
See also	`RegisterCallback()`		

Methods

SetPlayState			
XAresult (*SetPlayState) (XAPlayItf self, XAuint32 state);			
Description	Requests a transition of the player into the given play state.		
Pre-conditions	None. The player may be in any state.		
Parameters	`self`	[in]	Interface self-reference.
	`state`	[in]	Desired playback state.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PERMISSION_DENIED`
	`XA_RESULT_CONTENT_CORRUPTED`
	`XA_RESULT_CONTENT_UNSUPPORTED`		

Comments	All state transitions are legal. The state defaults to `XA_PLAYSTATE_STOPPED`. Note that although the state change is immediate, there may be some latency between the execution of this method and its effect on behavior. In this sense, a player's state technically represents the application's intentions for the player. Note that the player's state has an effect on the player's prefetch status (see `XAPrefetchStatusItf` for details). The player may return `XA_RESULT_PERMISSION_DENIED, XA_RESULT_CONTENT_CORRUPTED` or `XA_RESULT_CONTENT_UNSUPPORTED` respectively if, at the time a state change is requested, it detects insufficient permissions, corrupted content, or unsupported content. When the player reaches the end of content, the play state will transition to paused and the play cursor will remain at the end of content.		

GetPlayState			
XAresult (*GetPlayState) (**XAPlayItf self,** **XAuint32 * pState** **);**			
Description	Gets the player's current play state.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pState`	[out]	Pointer to a location to receive the current play state of the player. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		

GetDuration			
XAresult (*GetDuration) (**XAPlayItf self,** **XAmillisecond * pMsec** **);**			
Description	Gets the duration of the current content, in milliseconds.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pMsec`	[out]	Pointer to a location to receive the number of milliseconds corresponding to the total duration of this current content. If the duration is unknown, this value shall be `XA_TIME_UNKNOWN`. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		

GetPosition

```
XAresult (*GetPosition) (
    XAPlayItf self,
    XAmillisecond * pMsec
);
```

Description	Returns the current position of the playback head relative to the beginning of the content.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pMsec`	[out]	Pointer to a location to receive the position of the playback head relative to the beginning of the content, and is expressed in milliseconds. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The returned value is bounded between 0 and the duration of the content. Note that the position is defined relative to the content playing at 1x forward rate; positions do not scale with changes in playback rate.		

RegisterCallback

```
XAresult (*RegisterCallback) (
    XAPlayItf self,
    xaPlayCallback callback,
    void * pContext
);
```

Description	Sets the playback callback function.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Callback function invoked when one of the specified events occurs. A `NULL` value indicates that there is no callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The callback function defaults to `NULL`. The context pointer can be used by the application to pass state to the callback function.		

SetCallbackEventsMask			
XAresult (*SetCallbackEventsMask) (XAPlayItf self, XAuint32 eventFlags);			
Description	Enables/disables notification of playback events.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`eventFlags`	[in]	Bitmask of play event flags indicating which callback events are enabled. The presence of a flag enables notification for the corresponding event. The absence of a flag disables notification for the corresponding event. See `XA_PLAYEVENT` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The callback event flags default to all flags cleared.		

GetCallbackEventsMask			
XAresult (*GetCallbackEventsMask) (XAPlayItf self, XAuint32 * pEventFlags);			
Description	Queries for the notification state (enabled/disabled) of playback events.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pEventFlags`	[out]	Pointer to a location to receive the bitmask of play event flags indicating which callback events are enabled. This must be non-`NULL`. See `XA_PLAYEVENT` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetMarkerPosition

```
XAresult (*SetMarkerPosition) (
    XAPlayItf self,
    XAmillisecond mSec
);
```

Description	Sets the position of the playback marker.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`mSec`	[in]	Position of the marker expressed in milliseconds and relative to the beginning of the content.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The player will notify the application when the playback head passes through the marker via a callback with a `XA_PLAYEVENT_HEADATMARKER` event. By default, there is no marker position defined. When a marker position coincides with a periodic position update (as specified by `SetPositionUpdatePeriod`), then both the marker position callback and the periodic position update callback shall be posted next to each other. The order of the two callbacks is insignificant.		
See Also	`ClearMarkerPosition(), SetPositionUpdatePeriod()`		

ClearMarkerPosition

```
XAresult (*ClearMarkerPosition) (
    XAPlayItf self
);
```

Description	Clears marker.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	This function succeeds even if the marker is already clear.		
See Also	`SetMarkerPosition()`		

GetMarkerPosition			
XAresult (*GetMarkerPosition) (**XAPlayItf self,** **XAmillisecond * pMsec** **);**			
Description	Queries the position of playback marker.		
Pre-conditions	A marker has been set (using `SetMarkerPosition()` with no intervening `ClearMarkerPosition()`).		
Parameters	`self`	[in]	Interface self-reference.
	`pMsec`	[out]	Pointer to a location to receive the position of the marker expressed in milliseconds, relative to the beginning of the content.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	None		
See Also	`SetMarkerPosition()`, `ClearMarkerPosition()`		

SetPositionUpdatePeriod			
XAresult (*SetPositionUpdatePeriod) (**XAPlayItf self,** **XAmillisecond mSec** **);**			
Description	Sets the interval between periodic position notifications.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`mSec`	[in]	Period between position notifications in milliseconds.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The player will notify the application when the playback head passes through the positions implied by the specified period. Those positions are defined as the whole multiples of the period relative to the beginning of the content. By default, the update period is 1000 milliseconds. When a periodic position update coincides with a marker position (as specified by `SetMarkerPosition`), then both the update period callback and the marker position callback shall be posted next to each other. The order of the two callbacks is insignificant.		
See Also	`SetMarkerPosition()`		

GetPositionUpdatePeriod			
XAresult (*GetPositionUpdatePeriod) (**XAPlayItf self,** **XAmillisecond * pMsec** **);**			
Description	Queries the interval between periodic position notifications.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMsec`	[out]	Pointer to a location to receive the period between position notifications in milliseconds. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

8.28 XAPlaybackRateItf

Description

`XAPlaybackRateItf` is an interface for controlling setting and retrieving the rate at which an object presents data. Rates are expressed as a permille type (namely, parts per thousand):

- Negative values indicate reverse presentation.
- A value of 0 indicates paused presentation.
- Positive values less than 1000 indicate slow forward rates.
- A value of 1000 indicates normal 1X forward playback.
- Positive values greater than 1000 indicate fast forward rates.

Defaults

The rate value defaults to 1000 (that is, normal 1X forward playback).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_PLAYBACKRATE;

struct XAPlaybackRateItf_;
typedef const struct XAPlaybackRateItf_ * const * XAPlaybackRateItf;

struct XAPlaybackRateItf_ {
    XAresult (*SetRate) (
        XAPlaybackRateItf self,
        XApermille rate
    );
    XAresult (*GetRate) (
        XAPlaybackRateItf self,
        XApermille * pRate
    );
    XAresult (*SetPropertyConstraints) (
        XAPlaybackRateItf self,
        XAuint32 constraints
    );
    XAresult (*GetProperties) (
        XAPlaybackRateItf self,
        XAuint32 * pProperties
    );
    XAresult (*GetCapabilitiesOfRate) (
        XAPlaybackRateItf self,
        XApermille rate,
        XAuint32 * pCapabilities
    );
```

```
    XAresult (*GetRateRange) (
        XAPlaybackRateItf self,
        XAuint8 index,
        XApermille * pMinRate,
        XApermille * pMaxRate,
        XApermille * pStepSize,
        XAuint32 * pCapabilities
    );
};
```

Interface ID

c36f1440-f776-11db-ac48-0002a5d5c51b

Methods

SetRate			
`XAresult (*SetRate) ( XAPlaybackRateItf self, XApermille rate );`			
Description	Sets the rate of presentation.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`rate`	[in]	Desired rate.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	1000 is the default rate. The application may query supported rates via the `getRateRange()` method. The `XA_RESULT_FEATURE_UNSUPPORTED` return value accommodates the circumstance where the content being played does not afford adjustments of the playback rate.		

GetRate			
XAresult (*GetRate) (**XAPlaybackRateItf self,** **XApermille * pRate** **);**			
Description	Gets the rate of presentation.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pRate`	[out]	Pointer to a location to receive the rate of the player. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetPropertyConstraints			
XAresult (*SetPropertyConstraints) (**XAPlaybackRateItf self,** **XAuint32 constraints** **);**			
Description	Sets the current rate property constraints.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`constraints`	[in]	Bitmask of the allowed rate properties requested. An implementation may choose any of the given properties to implement rate and none of the excluded properties. See `XA_RATEPROP` macros. All video properties (and their corresponding bits) are mutually exclusive. All audio properties (and their corresponding bits) are mutually exclusive. If the bitmask is not well-formed,,this method returns `XA_RESULT_PARAMETER_INVALID`. If the constraints cannot be satisfied, this method returns `XA_RESULT_FEATURE_UNSUPPORTED`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Note that rate property capabilities may vary from one rate to another. This implies that a setting supported for one rate may be unsupported for another. Implementations controlling the rate of both video and audio will always have exactly one video property set (exactly one bit in the least significant byte) and one audio property set (exactly one bit in the second least significant byte). Implementations controlling the rate only of audio will have only an audio property set. The default video and audio properties are `XA_RATEPROP_SMOOTHVIDEO` and `XA_RATEPROP_NOPITCHCORAUDIO,` respectively.		

GetProperties			
XAresult (*GetProperties) (**XAPlaybackRateItf self,** **XAuint32 * pProperties** **);**			
Description	Gets the current properties.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pProperties`	[out]	Pointer to a location to receive the bitmask expressing the current rate properties. This must be non-`NULL`. See `XA_RATEPROP` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetCapabilitiesOfRate			
XAresult (*GetCapabilitiesOfRate) (**XAPlaybackRateItf self,** **XApermille rate,** **XAuint32 * pCapabilities** **);**			
Description	Gets the capabilities of the specified rate.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`rate`	[in]	Rate for which the capabilities are being queried.
	`pCapabilities`	[out]	Pointer to a location to receive the bitmask expressing capabilities of the given rate in terms of rate properties. See `XA_RATEPROP` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	An application may also leverage this method to verify that a particular rate is supported.		

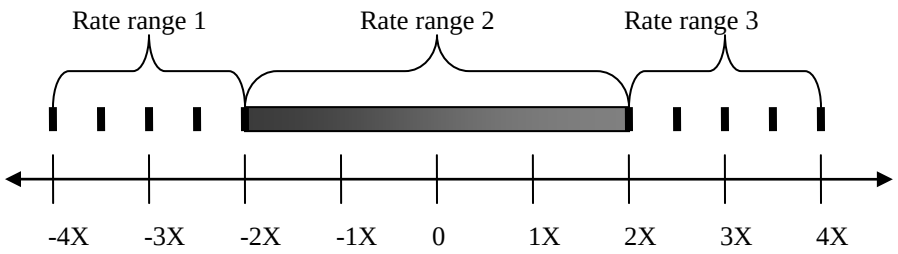
GetRateRange

```
XAresult (*GetRateRange) (
    XAPlaybackRateItf self,
    XAuint8 index,
    XApermille * pMinRate,
    XApermille * pMaxRate,
    XApermille * pStepSize,
    XAuint32 * pCapabilities
);
```

Description	Retrieves the ranges of rates supported.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`index`	[in]	Index of the range being queried. If an implementation supports n rate ranges, this value is between 0 and (n-1) and all values greater than n cause the method to return `XA_RESULT_PARAMETER_INVALID`.
	`pMinRate`	[out]	Pointer to a location to receive the minimum rate supported. May be negative or positive. Must be equal to or less than maxRate. This must be non-`NULL`.
	`pMaxRate`	[out]	Pointer to a location to receive the maximum rate supported. May be negative or positive. Must be equal to or greater than minRate. This must be non-`NULL`.
	`pStepSize`	[out]	Pointer to a location to receive the distance between one rate and an adjacent rate in the range. A value of zero denotes a continuous range. This must be non-`NULL`.
	`pCapabilities`	[out]	Pointer to a location to receive the bitmask of supported rate properties in the given range. This must be non-`NULL`. See `XA_RATEPROP` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	An implementation expresses the set of supported rates as one or more ranges. Each range is defined by the lowest and highest rates in the range, the step size between these bounds, and the rate properties of this range. If all rates an implementation supports are evenly spaced and have same capabilities, `GetRateRange()` method may return a single range. If not, the `GetRateRange()` method will return as many ranges as necessary in order to adequately express the set of rates (and associated properties) supported. In this case, the application must call `GetRateRange()` multiple times to query all the ranges; `GetRateRange()` returns only one range per call.		



Rate range examples: Range 1 has a min of -4000, a max of -2000 and a step of 500. Range 2 has a min of -2000, a max of 2000, and a step of 0. Range 3 has a min of 2000, a max of 4000 and a step of 500.

Figure 17: Example rate ranges

8.29 XAPrefetchStatusItf

Description

`XAPrefetchStatusItf` is an interface for querying the prefetch status of a player.

The prefetch status is a continuum ranging from no data prefetched to the maximum amount of data prefetched. It includes a range where underflow may occur and a range where there is a sufficient amount of data present. The underflow and sufficient data ranges may not relate to fixed fill level positions, but be implementation dependent and dynamically vary based on factors as e.g. buffering length, consumption rate, communication latency, hysteresis, etc. The prefetch status interface allows an application to query for prefetch status or register prefetch status callbacks. The latency of status and fill level callbacks are implementation dependent.

One example usage of the `XAPrefetchStatusItf` is to order the player into paused state when receiving an underflow event and into play state when receiving a sufficient data event when playing network stored media sources. Another example usage is to display fill level percentage to the end user by using the callback and the `GetFillLevel` method.

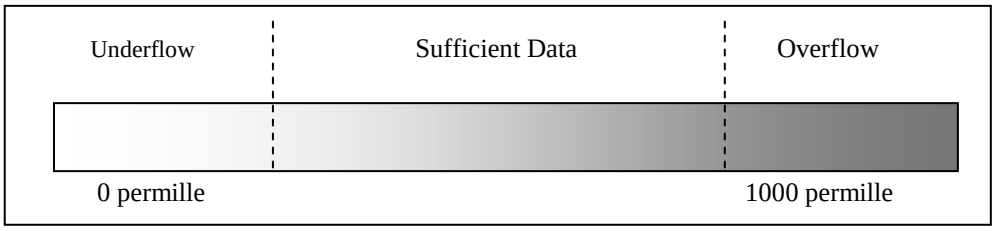


Figure 18: Prefetch continuum range

This interface is an implicit interface of Media Player objects (see section 7.4).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_PREFETCHSTATUS;

struct XAPrefetchStatusItf_;
typedef const struct XAPrefetchStatusItf_
    * const * XAPrefetchStatusItf;

struct XAPrefetchStatusItf_ {
    XAresult (*GetPrefetchStatus) (
        XAPrefetchStatusItf self,
        XAuint32 * pStatus
    );
    XAresult (*GetFillLevel) (
        XAPrefetchStatusItf self,
        XApermille * pLevel
    );
```

```
    XAresult (*RegisterCallback) (
        XAPrefetchStatusItf self,
        xaPrefetchCallback callback,
        void * pContext
    );
    XAresult (*SetCallbackEventsMask) (
        XAPrefetchStatusItf self,
        XAuint32 eventFlags
    );
    XAresult (*GetCallbackEventsMask) (
        XAPrefetchStatusItf self,
        XAuint32 * pEventFlags
    );
    XAresult (*SetFillUpdatePeriod) (
        XAPrefetchStatusItf self,
        XApermille period
    );
    XAresult (*GetFillUpdatePeriod) (
        XAPrefetchStatusItf self,
        XApermille * pPeriod
    );
};
```

Interface ID

cceac0a0-f776-11db-bb9c-0002a5d5c51b

Defaults

Initially, there is no callback registered, the fill update period is 100 permille, and the event flags are clear.

Callbacks

xaPrefetchCallback

```
typedef void (XAAPIENTRY * xaPrefetchCallback) (
    XAPrefetchStatusItf caller,
    void * pContext,
    XAuint32 event
);
```

Description	Notifies the player application of a prefetch event.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`event`	[in]	Event that has occurred. See `XA_PREFETCHEVENT` macros in section 9.2.55.
Comments	None		
See also	`RegisterCallback()`		

Methods

GetPrefetchStatus			
XAresult (*GetPrefetchStatus) (**XAPrefetchStatusItf self,** **XAuint32 * pStatus** **);**			
Description	Gets the player's current prefetch status.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pStatus`	[out]	Pointer to a location to receive the current prefetch status of the player. The status returned is of the XA_PREFETCHSTATUS defines, see section 9.2.56. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetFillLevel			
XAresult (*GetFillLevel) (**XAPrefetchStatusItf self,** **XApermille * pLevel** **);**			
Description	Queries the fill level of the prefetch.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pLevel`	[out]	Pointer to a location to receive the data fill level in parts per thousand. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The fill level is not tied to specific buffer within a player, but indicates more abstractly the progress a player has made in preparing data for playback.		

RegisterCallback			
XAresult (*RegisterCallback) (XAPrefetchStatusItf self, xaPrefetchCallback callback, void * pContext);			
Description	Sets the prefetch callback function.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Callback function invoked when one of the specified events occurs. A `NULL` value indicates that there is no callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Callback function defaults to `NULL`. The context pointer can be used by the application to pass state to the callback function.		
See Also	XA_PREFETCHEVENT macros (see section 9.2.55)		

SetCallbackEventsMask			
XAresult (*SetCallbackEventsMask) (XAPrefetchStatusItf self, XAuint32 eventFlags);			
Description	Sets the notification state of the prefetch events.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`eventFlags`	[in]	Bitmask of prefetch event flags indicating which callback events are enabled. See `XA_PREFETCHEVENT` macros in section 9.2.55.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Event flags default to all flags cleared.		

GetCallbackEventsMask			
XAresult (*GetCallbackEventsMask) (**XAPrefetchStatusItf self,** **XAuint32 * pEventFlags** **);**			
Description	Queries the notification state of the prefetch events.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pEventFlags`	[out]	Pointer to a location to receive the bitmask of prefetch event flags indicating which callback events are enabled. This must be non-`NULL`. See `XA_PREFETCHEVENT` macros, see section 9.2.55.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetFillUpdatePeriod			
XAresult (*SetFillUpdatePeriod) (**XAPrefetchStatusItf self,** **XApermille period** **);**			
Description	Sets the notification period for fill level updates. This period implies the set discrete fill level values that will generate notifications from the player.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`period`	[in]	Non-zero period between fill level notifications in permille. Notifications will occur at 0 permille (i.e. empty) and at whole number increments of the period from 0. For instance, if the period is 200 permille (i.e. 20%), then the player will generate a notification when 0%, 20%, 40%, 60%, 80%, or 100% full. The default period is 100 permille.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetFillUpdatePeriod			
XAresult (*GetFillUpdatePeriod) (**XAPrefetchStatusItf self,** **XApermille * pPeriod** **);**			
Description	Queries the notification period for fill level updates.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pPeriod`	[out]	Pointer to a location to receive the period between fill level notifications in permille. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

8.30 XARadioItf

Description

XARadioItf is for controlling the basic functionality of the analog audio radio. The interface contains methods for selecting the frequency range and modulation used by the tuner (SetFreqRange()), for tuning to a certain frequency manually (SetFrequency()) or by automatically seeking (Seek()), for using the tuner presets of the device and for accessing squelching functionality, stereo mode and the signal strength.

Please note that Hertz (Hz) is used as the unit of the frequency. For example, 100,000,000 Hz equals 100.0 MHz.

This interface an implicit interface of Radio I/O device objects, see section 7.8. See section D.3 for an example using this interface.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_RADIO;

struct XARadioItf_;
typedef const struct XARadioItf_ * const * XARadioItf;

struct XARadioItf_ {
    XAresult (*SetFreqRange) (
        XARadioItf self,
        XAuint8 range
    );
    XAresult (*GetFreqRange) (
        XARadioItf self,
        XAuint8 * pRange
    );
    XAresult (*IsFreqRangeSupported) (
        XARadioItf self,
        XAuint8 range,
        XAboolean * pSupported
    );
    XAresult (*GetFreqRangeProperties) (
        XARadioItf self,
        XAuint8 range,
        XAuint32 * pMinFreq,
        XAuint32 * pMaxFreq,
        XAuint32 * pFreqInterval
    );
    XAresult (*SetFrequency) (
        XARadioItf self,
        XAuint32 freq
    );
    XAresult (*CancelSetFrequency) (
        XARadioItf self
    );
    XAresult (*GetFrequency) (
        XARadioItf self,
        XAuint32 * pFreq
    );
```

```
    XAresult (*SetSquelch) (
        XARadioItf self,
        XAboolean squelch
    );
    XAresult (*GetSquelch) (
        XARadioItf self,
        XAboolean * pSquelch
    );
    XAresult (*SetStereoMode) (
        XARadioItf self,
        XAuint32 mode
    );
    XAresult (*GetStereoMode) (
        XARadioItf self,
        XAuint32 * pMode
    );
    XAresult (*GetSignalStrength) (
        XARadioItf self,
        XAuint32 * pStrength
    );
    XAresult (*Seek) (
        XARadioItf self,
        XAboolean upwards
    );
    XAresult (*StopSeeking) (
        XARadioItf self
    );
    XAresult (*GetNumberOfPresets) (
        XARadioItf self,
        XAuint32 * pNumPresets
    );
    XAresult (*SetPreset) (
        XARadioItf self,
        XAuint32 preset,
        XAuint32 freq,
        XAuint8 range,
        XAuint32 mode,
        const XAchar * pName
    );
    XAresult (*GetPreset) (
        XARadioItf self,
        XAuint32 preset,
        XAuint32 * pFreq,
        XAuint8 * pRange,
        XAuint32 * pMode,
        XAchar * pName,
        XAuint16 * pNameLength
    );
    XAresult (*RegisterRadioCallback) (
        XARadioItf self,
        xaRadioCallback callback,
        void * pContext
    );
};
```

Interface ID

b316ad80-df05-11db-b5b6-0002a5d5c51b

Defaults

No callback registered.

Callbacks

xaRadioCallback			
`typedef void (XAAPIENTRY * xaRadioCallback) ( XARadioItf caller, void * pContext, XAuint32 event, XAuint32 eventIntData, XAboolean eventBooleanData );`			
Description	Notifies the application about radio event.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`event`	[in]	One of the radio event codes, see section 9.2.58 `XA_RADIO_EVENT` macros.
	`eventIntData`	[in]	Event specific integer parameter. Specifies additional notification callback event specific information. The contents of this parameter are dependent on the event being reported. See section 9.2.58 `XA_RADIO_EVENT` macros.
	eventBooleanData	[in]	Event specific Boolean argument. Specifies additional notification callback event specific information. The contents of this parameter are dependent on the event being reported. See section 9.2.58 `XA_RADIO_EVENT` macros.
Comments	None		
See Also	None		

Methods

SetFreqRange			
XAresult (*SetFreqRange) (**XARadioItf self,** **XAuint8 range** **);**			
Description	Sets the frequency range. Asynchronous – `xaRadioCallback()` callback with `XA_RADIO_EVENT_FREQUENCY_RANGE_CHANGED or XA_RADIO_EVENT_FREQUENCY_RANGE_ERROR` events used for notification of the result.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`range`	[in]	New frequency range. See `XA_FREQRANGE` macros section for ranges. Use `IsFreqRangeSupported()` to query supported ranges.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	See Section 9.2.58 `XA_RADIO_EVENT` macros.		

GetFreqRange			
XAresult (*GetFreqRange) (**XARadioItf self,** **XAuint8 * pRange** **);**			
Description	Gets the current frequency range.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pRange`	[out]	Current frequency range. See `XA_FREQRANGE` macros section for ranges.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

IsFreqRangeSupported			
XAresult (*IsFreqRangeSupported) (**XARadioItf self,** **XAuint8 range,** **XAboolean * pSupported** **);**			
Description	Queries if the given frequency range is supported.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`range`	[out]	Frequency range whose availability is queried. See `XA_FREQRANGE` macro section for ranges.
	`pSupported`	[out]	True if the range is supported, false otherwise.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetFreqRangeProperties

```
XAresult (*GetFreqRangeProperties) (
    XARadioItf self,
    XAuint8 range,
    XAuint32 * pMinFreq,
    XAuint32 * pMaxFreq,
    XAuint32 * pFreqInterval
);
```

Description	Returns the minimum and maximum supported frequencies and the modulation of the given frequency range.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`range`	[in]	Frequency range whose properties are queried. See `XA_FREQRANGE` macros section for ranges. Use IsFreqRangeSupported to query supported ranges first.
	`pMinFreq`	[out]	Minimum frequency of the given frequency range in Hertz.
	`pMaxFreq`	[out]	Maximum frequency of the given frequency range in Hertz.
	`pFreqInterval`	[out]	Interval between supported frequencies on the given frequency range. That is, the frequency accuracy of the device.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetFrequency			
XAresult (*SetFrequency) (XARadioItf self, XAuint32 freq);			
Description	Sets the frequency asynchronously – `xaRadioCallback()` callback with `XA_RADIO_EVENT_FREQUENCY_CHANGED` or XA_RADIO_EVENT_FREQUENCY_ERROR events used for notification of the result. The implementation rounds the given value to the nearest supported one. See `pFreqInterval` parameter of `GetFreqRangeProperties()` method.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`freq`	[in]	New frequency in Hertz. Must be between `pMinFreq` and `pMaxFreq` parameters of `GetFreqRangeProperties` method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	See Section 9.2.58 `XA_RADIO_EVENT` macros.		

CancelSetFrequency			
XAresult (*CancelSetFrequency) (XARadioItf self);			
Description	Cancels an outstanding `SetFrequency()` request. The method blocks while canceling the outstanding request. Has not effect if no set frequency operation is ongoing.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
Return value	The return value can be the following: `XA_RESULT_SUCCESS`		
Comments	None		

GetFrequency			
XAresult (*GetFrequency) (**XARadioItf self,** **XAuint32 * pnFreq** **);**			
Description	Gets the current frequency.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pnFreq`	[out]	Current frequency in Hertz.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetSquelch			
XAresult (*SetSquelch) (**XARadioItf self,** **XAboolean squelch** **);**			
Description	Toggles the squelch (muting in frequencies without broadcast).		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`squelch`	[in]	True to switch on squelch and false to switch it off.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	Squelch changes made by other applications can trigger a XA_RADIO_EVENT_SQUELCH_CHANGED event. See Section 9.2.58 `XA_RADIO_EVENT` macros.		

GetSquelch			
XAresult (*GetSquelch) (**XARadioItf self,** **XAboolean * pSquelch** **);**			
Description	Queries the squelch setting (muting in frequencies without broadcast).		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pSquelch`	[out]	True when squelch is on and false if it is off.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	Squelch changes made by other applications can trigger a XA_RADIO_EVENT_SQUELCH_CHANGED event. See Section 9.2.58 `XA_RADIO_EVENT` macros.		

SetStereoMode			
XAresult (*SetStereoMode) (**XARadioItf self,** **XAuint32 mode** **);**			
Description	Sets the current stereo mode.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`mode`	[in]	New stereo mode. See `XA_STEREO_MODE` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	`XA_RESULT_FEATURE_UNSUPPORTED` is returned if the given mode is not supported for the current content. The supported modes are dependent on the broadcast content and some modes cannot be selected if not appropriate. For example, `XA_STEREOMODE_STEREO` mode is not possible if the broadcast is mono.		

GetStereoMode			
XAresult (*GetStereoMode) (**XARadioItf self,** **XAuint32 * pMode** **);**			
Description	Queries the current stereo mode.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMode`	[out]	Current stereo mode. See `XA_STEREO_MODE` macros.

Return value	The return value can be the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetSignalStrength			
XAresult (*GetSignalStrength) (**XARadioItf self,** **XAuint32 * pStrength** **);**			
Description	Returns the signal strength in per cents.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pStrength`	[out]	Signal strength in per cents.

Return value	The return value can be the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

Seek			
XAresult (*Seek) (**XARadioItf self,** **XAboolean upwards** **);**			
Description	Starts the seek from the current frequency to the given direction. Asynchronous – `xaRadioCallback()` callback with `XA_RADIO_EVENT_SEEK_COMPLETED` event is used for notifying of the result. If the end of the tuner's frequency band is reached before a signal was found, the scan continues from the other end until a signal is found or the original frequency is reached.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`upwards`	[in]	If true the seek progresses towards higher frequencies and if false the seek progresses towards lower frequencies.
Return value	The return value can be the following: `XA_RESULT_SUCCESS`		
Comments	None		

StopSeeking			
XAresult (*StopSeeking) (**XARadioItf self** **);**			
Description	Cancels an outstanding seek request. The method blocks while canceling the outstanding request. After cancellation, the frequency is the one where seeking stopped. Has not effect if no seek operation is ongoing.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
Return value	The return value can be the following: `XA_RESULT_SUCCESS`		
Comments	None		

GetNumberOfPresets			
XAresult (*GetNumberOfPresets) (**XARadioItf self,** **XAuint32 * pNumPresets** **);**			
Description	Returns the number of preset slots the device has for storing the presets.		
Pre-conditions	None		
Parameters	self	[in]	Interface self-reference.
	pNumPresets	[out]	Number of presets.
Return value	The return value can be one of the following: XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID		
Comments	The presets are persistent and therefore all the preset slots contain always values. Please note that in some devices it is possible to change the preset values also by using other mechanism than OpenMAX AL. The initial values in a brand new device are implementation dependent.		

SetPreset			
XAresult (*SetPreset) (**XARadioItf self,** **XAuint32 preset,** **XAuint32 freq,** **XAuint8 range,** **XAuint32 mode,** **const XAchar * pName** **);**			
Description	Sets the preset.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`preset`	[in]	Index number of the preset. Should be between 1 and the amount of presets (returned by `GetNumberOfPresets()`).
	`freq`	[in]	Frequency to be stored to the preset. Use `GetFreqRangeProperties()` to query supported ranges first.
	`range`	[in]	Frequency range to be stored to the preset. See `XA_FREQRANGE` macros section for ranges. Use `IsFreqRangeSupported()` to query supported ranges first.
	`mode`	[in]	Stereo mode to be stored to the preset.
	`pName`	[in]	Name for the preset.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The presets are persistent and therefore all the preset slots contain always values. Please note that in some devices it is possible to change the preset values also by using other mechanism than OpenMAX AL. The initial values in a brand new device are implementation dependent.		

GetPreset

```
XAresult (*GetPreset) (
    XARadioItf self,
    XAuint32 preset,
    XAuint32 * freq,
    XAuint8 * range,
    XAuint32 * mode,
    XAchar * pName,
    XAuint16 * pNameLength
);
```

Description	Gets the settings stored into a preset.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`preset`	[in]	Index number of the preset. Should be between 1 and the amount of presets (returned by `GetNumberOfPresets()`).
	`freq`	[out]	Frequency in Hertz stored to the preset.
	`range`	[out]	Frequency range stored to the preset. See `XA_FREQRANGE` macros section for ranges.
	`mode`	[out]	Stereo mode stored to the preset.
	`pName`	[out]	Name of the preset. If this parameter is NULL the required length of the buffer is returned in the `pNameLength` parameter.
	`pNameLength`	[in/out]	As an output, specifies the length of the name including the terminating NULL. As an input, specifies the length of the given pName char array (ignored if `pName` is NULL).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	The presets are persistent and therefore all the preset slots contain always values. Please note that in some devices it is possible to change the preset values also by using other mechanism than OpenMAX AL. The initial values in a brand new device are implementation dependent. If the char array passed in the pName parameter is of insufficient length the pName parameter is filled to its maximum, the `pNameLength` parameter is updated to the needed length and a `XA_RESULT_BUFFER_INSUFFICIENT` return value is returned. The returned string is always valid. That is, the null-terminator always exists and multibyte characters are not cut in the middle.		

RegisterRadioCallback			
XAresult (*RegisterRadioCallback) (**XARadioItf self,** **xaRadioCallback callback,** **void * pContext** **);**			
Description	Sets or clears the xaRadioCallback.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

8.31 XARDSItf

Description

This interface is for accessing the Radio Data System for VHF/FM sound broadcasting (RDS) (IEC 62106) features. This interface can also be used with Radio Broadcast Data System (RBDS) (United States RBDS Standard, NRSC-4-A), but RDS terminology (Glossary of RDS Terms) is used in this API documentation.

This interface can be exposed on the Radio I/O device object, if RDS is supported.

Please note that right after setting a new frequency, the RDS fields might contain empty or default values and it can take (in order of seconds) time until RDS data is received; these callbacks will be called then.

See section D.3 for an example using this interface.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_RDS;

struct XARDSItf_;
typedef const struct XARDSItf_ * const * XARDSItf;

struct XARDSItf_ {
    XAresult (*QueryRDSSignal) (
        XARDSItf self,
        XAboolean * pIsSignal
    );
    XAresult (*GetProgrammeServiceName) (
        XARDSItf self,
        XAchar * pPSname
    );
    XAresult (*GetRadioText) (
        XARDSItf self,
        XAchar * pRt
    );
    XAresult (*GetRadioTextPlus) (
        XARDSItf self,
        XAuint8 contentType,
        XAchar * pInformationElement,
        XAchar * pDescriptor,
        XAuint8 * pDescriptorContentType
    );
    XAresult (*GetProgrammeType) (
        XARDSItf self,
        XAuint32 * pPty
    );
    XAresult (*GetProgrammeTypeString) (
        XARDSItf self,
        XAboolean isLengthMax16,
        XAchar * pPty
    );
```

```
XAresult (*GetProgrammeIdentificationCode) (
    XARDSItf self,
    XAint16 * pPICode
);
XAresult (*GetClockTime) (
    XARDSItf self,
    XAtime * pDateAndTime
);
XAresult (*GetTrafficAnnouncement) (
    XARDSItf self,
    XAboolean * pTa
);
XAresult (*GetTrafficProgramme) (
    XARDSItf self,
    XAboolean * pTp
);
XAresult (*SeekByProgrammeType) (
    XARDSItf self,
    XAuint32 pty,
    XAboolean upwards
);
XAresult (*SeekTrafficAnnouncement) (
    XARDSItf self,
    XAboolean upwards
);
XAresult (*SeekTrafficProgramme) (
    XARDSItf self,
    XAboolean upwards
);
XAresult (*SetAutomaticSwitching) (
    XARDSItf self,
    XAboolean automatic
);
XAresult (*GetAutomaticSwitching) (
    XARDSItf self,
    XAboolean * pAutomatic
);
XAresult (*SetAutomaticTrafficAnnouncement) (
    XARDSItf self,
    XAboolean automatic
);
XAresult (*GetAutomaticTrafficAnnouncement) (
    XARDSItf self,
    XAboolean * pAutomatic
);
XAresult (*GetODAGroup) (
    XARDSItf self,
    XAuint16 AID,
    xaGetODAGroupCallback callback,
    void * pContext
);
XAresult (*SubscribeODAGroup) (
    XARDSItf self,
    XAint16 group,
    XAboolean useErrorCorrection
);
```

```
    XAresult (*UnsubscribeODAGroup) (
        XARDSItf self,
        XAint16 group
    );
    XAresult (*ListODAGroupSubscriptions) (
        XARDSItf self,
        XAint16* pGroups,
        XAuint32* pLength
    );
    XAresult (*RegisterRDSCallback) (
        XARDSItf self,
        xaRDSCallback callback,
        void * pContext
    );
    XAresult (*RegisterODADataCallback) (
        XARDSItf self,
        xaNewODADataCallback callback,
        void * pContext
    );
};
```

Interface ID

2e79e200-d71c-11df-9aa6-0002a5d5c51b

Defaults

No callback registered.

Callbacks

xaGetODAGroupCallback			
typedef void (XAAPIENTRY * xaGetODAGroupCallback) (XARadioItf caller, void * pContext, XAboolean success, XAint16 group, XAuint16 message);			
Description	Callback of the `XARDSItf::GetODAGroup()` method. Gives asynchronously the application Group and the message bits concerning the given ODA (Open Data Application).		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`success`	[in]	True if the query was successful; false if there is no data available with the given ODA Application ID.
	`group`	[in]	Group. (0A=0, 0B=1, 1A=2, 1B=3...). -1 if there is no data available with the given ODA Application ID.
	`message`	[in]	Message bits of the given ODA. (16 bits)
Comments	If the ODA uses both type A and B groups, the callback will be executed twice, once for each group type.		
See Also	None		

xaNewODADataCallback			
typedef void (XAAPIENTRY * xaNewODADataCallback) (XARDSItf caller, void * pContext, XAint16 group, XAuint64 data);			
Description	New data from a subscribed ODA group has been received.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`group`	[in]	Group. (0A=0, 0B=1, 1A=2, 1B=3...)
	`data`	[in]	Payload data. (37 (least significant) bits for type A groups and 21 (least significant) bits for type B groups; the rest of the bits will be zeros)
Comments	None		
See Also	None		

xaRDSCallback			

`typedef void (XAAPIENTRY * xaRDSCallback) (`
`  XARDSItf caller,`
`  void * pContext,`
`  XAuint16 event,`
`  XAuint8 eventData`
`);`			

Description	This callback executes whenever at least one of the RDS fields changes. The application should use `XARDSItf` to query the new field values.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
	`event`	[in]	Bitwise OR of the RDS Event macros, which tells which of the RDS fields have changed their value.
	`eventData`	[in]	Event specific integer parameter. Specifies additional notification callback event specific information. The contents of this parameter are dependent on the event being reported. See 9.2.61 RDS Event macros.
Comments	Please note that after changing the frequency it might take a couple of seconds before all the RDS fields are received. Typically not all the fields are received at same time. Therefore, typically after changing the frequency, multiple `xaRDSCallback()` will take place, each containing only the events for the new fields received at time.		
See Also	Note: use `xaRadioCallback()` and `XA_RADIO_EVENT_SEEK_COMPLETED` callback from tuning callbacks also RDS-based seeks.		

Methods

QueryRDSSignal			

`XAresult (*QueryRDSSignal) (`
`  XARDSItf self,`
`  XAboolean * pIsSignal`
`);`			

Description	Returns the status of the RDS reception.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pIsSignal`	[out]	True if RDS signal is received, false otherwise.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetProgrammeServiceName			
XAresult (*GetProgrammeServiceName) (**XARDSItf self,** **XAchar * pPSname** **);**			
Description	Gets the current Programme Service name (PS).		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pPSname`	[out]	Name of the Programme Service or a zero-length string if unknown. The length of the name is 8 characters. Therefore, the application needs to pass here an array of length 17 (since each character used in RDS can take two UTF-8 encoded words and the string is null-terminated).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetRadioText			
XAresult (*GetRadioText) (**XARDSItf self,** **XAchar * pRt** **);**			
Description	Gets the current Radio Text (RT).		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pRt`	[out]	Radio Text or zero-length string if unknown. The length of the Radio Text is 64 characters maximum and it is null-terminated. Therefore, the application needs to pass here an array of length 129 (since each character used in RDS can take two UTF-8 encoded words and the string is null-terminated).

Return value	The return value can be the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetRadioTextPlus			
XAresult (*GetRadioTextPlus) (**XARDSItf self,** **XAuint8 contentType,** **XAchar * pInformationElement,** **XAchar * pDescriptor,** **XAuint8 * pDescriptorContentType** **);**			
Description	Gets the current Radio Text+ (RT+) information element based on the given class code.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`contentType`	[in]	Radio Text+ class code of the information element that is queried. See [9.2.64] `XA_RDSRTPLUS` macros.
	`pInformationElement`	[out]	Radio Text+ information element or zero-length string if unknown. The length of the Radio Text+ information element is 64 characters maximum. Therefore, the application needs to pass here an array of length 129 (since each character used in RDS can take two UTF-8 encoded words and the string is null-terminated).
	`pDescriptor`	[out]	Descriptor associated with the Radio Text+ information element that was queried. The length of the descriptor element is also 64 characters maximum. Therefore, the application needs to pass here an array of length 129 (since each character used in RDS can take two UTF-8 encoded words and the string is null-terminated). Should be ignored if p`DescriptorContentType` is zero.
	`pDescriptorContentType`	[out]	The Radio Text+ class of the descriptor. (One of the classes `XA_RDSRTPLUS_PLACE`, `XA_RDSRTPLUS_APPOINTMENT`, `XA_RDSRTPLUS_IDENTIFIER`, `XA_RDSRTPLUS_PURCHASE` or `XA_RDSRTPLUS_GETDATA`.) If this is zero, no associated descriptor is available for the queried RT+ information element.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	A RT+ information element can be complemented by another information element, p`Descriptor`. Therefore, this method has also p`Descriptor` and `pDescriptorContentType` as additional [out] parameters. The descriptor itself belongs to PLACE, APPOINTMENT, PURCHASE or GETDATA class.		

GetProgrammeType

```
XAresult (*GetProgrammeType) (
    XARDSItf self,
    XAuint32 * pPty
);
```

Description	Gets the current Programme TYpe (PTY) as short. The return value zero corresponds to No Programme Type or to undefined type. Please note that PTYs in RBDS differ from the ones in RDS.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pPty`	[out]	Programme TYpe or zero (`XA_RDSPTYNONE` or `XA_RBDSPTYNONE`) for undefined type.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetProgrammeTypeString

```
XAresult (*GetProgrammeTypeString) (
    XARDSItf self,
    XAboolean isLengthMax16,
    XAchar * pPty
);
```

Description	Gets the current Programme TYpe (PTY) as a String with the maximum of 8 or 16 characters in English (char set TBD) as defined in RDS and RBDS specifications. Please note that PTYs in RBDS differ from the ones in RDS.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`isLengthMax16`	[in]	True for the maximum length of 16 characters, false for the maximum length of 8 characters.
	`pty`	[out]	Programme TYpe or “None” for an undefined type. The application needs to pass here an array of length 17 or 33 depending on the isLengthMax16 parameter (since each character used in RDS can take two UTF-8 encoded words and the string is null-terminated).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetProgrammeIdentificationCode			
XAresult (*GetProgrammeIdentificationCode) (**XARDSItf self,** **XAint16 * pPIcode** **);**			
Description	Gets the current Programme Identification code (PI). The PI is not intended for directly displaying to the end user, but instead to identify uniquely a programme. This can be used to detect that two frequencies are transmitting the same programme.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pPIcode`	[out]	Programme Identification code or zero for an undefined PI code.

Return value	The return value can be the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetClockTime			
XAresult (*GetClockTime) (**XARDSItf self,** **XAtime * pDateAndTime** **);**			
Description	Gets the current Clock Time and date (CT).		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pDateAndTime`	[out]	Current time and date.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetTrafficAnnouncement			
XAresult (*GetTrafficAnnouncement) (**XARDSItf self,** **XAboolean * pTa** **);**			
Description	Gets the current status of the Traffic Announcement (TA) switch.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pTa`	[out]	True if TA is on, false otherwise

Return value	The return value can be the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetTrafficProgramme			
XAresult (*GetTrafficProgramme) (**XARDSItf self,** **XAboolean * pTp** **);**			
Description	Gets the current status of the Traffic Programme (TP) switch.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pTp`	[out]	True if TP is on, false otherwise

Return value	The return value can be the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SeekByProgrammeType			
XAresult (*SeekByProgrammeType) (**XARDSItf self,** **XAuint32 pty,** **XAboolean upwards** **);**			
Description	Seeks for the frequency sending the given Programme TYpe (PTY). If the end of the tuner's frequency band is reached before the given Programme TYpe is found, the scan continues from the other end until the given Programme TYpe is found or the original frequency is reached. Asynchronous - tuner callback `xaRadioCallback()` and `XA_RADIO_EVENT_SEEK_COMPLETED` is used for notifying of the result. `StopSeeking()` method of `XARadioItf` can be used to abort an ongoing seek.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pty`	[in]	Programme TYpe to seek for. `XA_RESULT_PARAMETER_INVALID` is returned if pty parameter is not following the RDS specification.
	`upwards`	[in]	If true the seek progresses towards higher frequencies and if false the seek progresses towards lower frequencies.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SeekTrafficAnnouncement

```
XAresult (*SeekTrafficAnnouncement) (
    XARDSItf self,
    XAboolean upwards
);
```

Description	Seeks for a frequency sending Traffic Announcement (TA). If the end of the tuner's frequency band is reached before a Traffic Announcement is found, the scan continues from the other end until a Traffic Announcement is found or the original frequency is reached. Asynchronous - tuner callback `xaRadioCallback()` and `XA_RADIO_EVENT_SEEK_COMPLETED` is used for notifying of the result. `StopSeeking()` method of `XARadioItf` can be used to abort an ongoing seek.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`upwards`	[in]	If true the seek progresses towards higher frequencies and if false the seek progresses towards lower frequencies.
Return value	The return value can be the following: `XA_RESULT_SUCCESS`		
Comments	None		

SeekTrafficProgramme

```
XAresult (*SeekTrafficProgramme) (
    XARDSItf self,
    XAboolean upwards
);
```

Description	Seeks for a frequency sending Traffic Programme (TP). If the end of the tuner's frequency band is reached before a Traffic Programme is found, the scan continues from the other end until a Traffic Programme is found or the original frequency is reached. Asynchronous - tuner callback `xaRadioCallback()` and `XA_RADIO_EVENT_SEEK_COMPLETED` is used for notifying of the result. `StopSeeking()` method of `XARadioItf` can be used to abort an ongoing seek.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`upwards`	[in]	If true the seek progresses towards higher frequencies and if false the seek progresses towards lower frequencies.
Return value	The return value can be the following: `XA_RESULT_SUCCESS`		
Comments	None		

SetAutomaticSwitching			
XAresult (*SetAutomaticSwitching) (**XARDSItf self,** **XAboolean automatic** **);**			
Description	Sets the automatic switching of the transmitter in the case of a stronger transmitter with the same PI presence. Based on AF and/or EON fields. Please note that NOT ALL IMPLEMENTATIONS SUPPORT THIS FUNCTIONALITY.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`automatic`	[in]	True to turn on the automatic switching, false to turn it off.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		

GetAutomaticSwitching			
XAresult (*GetAutomaticSwitching) (**XARDSItf self,** **XAboolean * pAutomatic** **);**			
Description	Gets the mode of the automatic switching of the transmitter in case of a stronger transmitter with the same PI presence.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pAutomatic`	[out]	True if the automatic switching is on, false otherwise.

Return value	The return value can be the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetAutomaticTrafficAnnouncement			
XAresult (*SetAutomaticTrafficAnnouncement) (**XARDSItf self,** **XAboolean automatic** **);**			
Description	Sets the automatic switching of the program in case of the presence of Traffic Announcement in another program. Based on TP and TA fields. Please note that NOT ALL IMPLEMENTATIONS SUPPORT THIS FUNCTIONALITY.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`automatic`	[in]	True to turn on the automatic switching, false to turn it off.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	None		

GetAutomaticTrafficAnnouncement			
XAresult (*GetAutomaticTrafficAnnouncement) (**XARDSItf self,** **XAboolean * pAutomatic** **);**			
Description	Gets the mode of the automatic switching of the program in case of the presence of Traffic Announcement in another program. Based on TP and TA fields.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pAutomatic`	[out]	True if the automatic switching is on, false otherwise.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

GetODAGroup			
XAresult (*GetODAGroup) (**XARDSItf self,** **XAuint16 AID,** **xaGetODAGroupCallback callback,** **void * pContext** **);**			
Description	Returns asynchronously via callback (`xaGetODAGroupCallback()`) the application Group and the message bits concerning the given ODA (Open Data Application). ODA is a mechanism that a broadcaster can use to transfer data that is not explicitly specified in the RDS standard. Open Data Applications are subject to a registration process. Transmission protocols used by ODAs may be public or private. See RDS Forum web page (http://www.rds.org.uk/)for details.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`AID`	[in]	ODA Application ID. (4 hex characters as short)
	`callback`	[in]	Address of the callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SubscribeODAGroup

```
XAresult (*SubscribeODAGroup) (
    XARDSItf self,
    XAint16 group,
    XAboolean useErrorCorrection
);
```

Description	Subscribes the given ODA group. If the given group was already subscribed, this call doesn't do anything. Only new data in groups that have been subscribed will cause a newODA callback.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`group`	[in]	Group to subscribe. (0A=0, 0B=1, 1A=2, 1B=3...)
	`useErrorCorrection`	[in]	True to use the following error correction: if the same data arrives twice within three data arrivals it is correct. False not to use error correction.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`GetODAGroup()` can be used to find out which group to subscribe, if the AID (Application Identification) is known.		

UnsubscribeODAGroup

```
XAresult (*UnsubscribeODAGroup) (
    XARDSItf self,
    XAint16 group
);
```

Description	Unsubscribes the given ODA group. If the given group has not been subscribed, this doesn't do anything. Only new data in groups that have been subscribed will cause a newODA callback.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`group`	[in]	Group to unsubscribe. (0A=0, 0B=1, 1A=2, 1B=3...)

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

ListODAGroupSubscriptions

```
XAresult (*ListODAGroupSubscriptions) (
    XARDSItf self,
    XAint16* pGroups,
    XAunit32* pLength
);
```

Description	Lists ODA groups that are currently subscribed.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pGroups`	[out]	An array of the groups that are subscribed. (0A=0, 0B=1, 1A=2, 1B=3...) The length of the needed array should be first figured out from `pLength` out parameter by calling this method with `pGroups` as null.
	`pLength`	[in/out]	As an output, specifies the length of the groups array. That is, the number of subscribed groups. As an input, specifies the length of the given `pGroups` array (ignored if `pGroups` is `NULL`).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	If the given length is smaller than the needed size XA_RESULT_BUFFER_INSUFFICIENT is returned and only data of the given size will be written.		

RegisterODADataCallback

```
XAresult (*RegisterODADataCallback) (
    XARDSItf self,
    xaNewODADataCallback callback,
    void * pContext
);
```

Description	Sets or clears the `xaNewODADataCallback()`. `xaNewODADataCallback()` is used tranfer the actual ODA data to the application.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

RegisterRDSCallback			
XAresult (*RegisterRDSCallback) (**XARDSItf self,** **xaRDSCallback callback,** **void * pContext** **);**			
Description	Sets or clears the `xaRDSCallback()`. `xaRDSCallback()` is used to monitor changes in RDS fields.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Address of the callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

8.32 XARecordItf

Description

`XARecordItf` is an interface for controlling the recording state of an object. The record state machine is as follows:

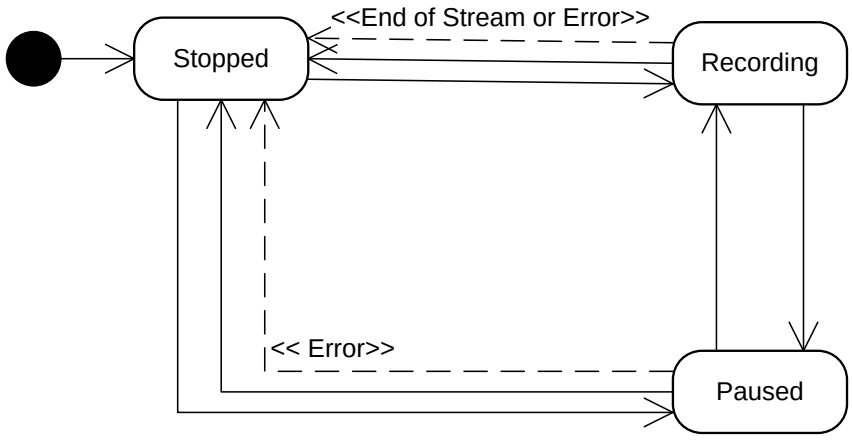


Figure 19: Record state machine

In case the disc gets full while recording to a file, XA_OBJECT_EVENT_RUNTIME_ERROR will be posted via xaObjectCallback with XA_RESULT_IO_ERROR as this callback's result parameter. The recorder will in that case autotransition into XA_RECORDSTATE_STOPPED state.

Table 13: Data Status and Recording State

Recording State	Destination[1] closed	Head[2] moving (sending data to destination)
Stopped	X	
Paused[3]		
Recording		X

[1] "Destination" denotes the sink of the recording process (for example, a file being written to).

[2] "Head" denotes the position of the recording process relative in time to the duration of the entire recording (for example, if the five seconds of video have been sent to the destination, the head is at five seconds).

[3] If a recorder transitions from Paused to Recording (without an intervening transition to Stopped), the newly captured data is appended to data already sent to the destination.

This interface is a mandated interface of Media Recorder objects (see section 7.5). See section D.4 for an example using this interface.

Prototype

```
XA_API extern const XAInterfaceID XA_IID_RECORD;

struct XARecordItf_;
typedef const struct XARecordItf_ * const * XARecordItf;

struct XARecordItf_ {
    XAresult (*SetRecordState) (
        XARecordItf self,
        XAuint32 state
    );
    XAresult (*GetRecordState) (
        XARecordItf self,
        XAuint32 * pState
    );
    XAresult (*SetDurationLimit) (
        XARecordItf self,
        XAmillisecond msec
    );
    XAresult (*GetPosition) (
        XARecordItf self,
        XAmillisecond * pMsec
    );
    XAresult (*RegisterCallback) (
        XARecordItf self,
        xaRecordCallback callback,
        void * pContext
    );
    XAresult (*SetCallbackEventsMask) (
        XARecordItf self,
        XAuint32 eventFlags
    );
    XAresult (*GetCallbackEventsMask) (
        XARecordItf self,
        XAuint32 * pEventFlags
    );
    XAresult (*SetMarkerPosition) (
        XARecordItf self,
        XAmillisecond mSec
    );
    XAresult (*ClearMarkerPosition) (
        XARecordItf self
    );
```

```
    XAresult (*GetMarkerPosition) (
        XARecordItf self,
        XAmillisecond * pMsec
    );
    XAresult (*SetPositionUpdatePeriod) (
        XARecordItf self,
        XAmillisecond mSec
    );
    XAresult (*GetPositionUpdatePeriod) (
        XARecordItf self,
        XAmillisecond * pMsec
    );
};
```

Interface ID

d7948cc0-f776-11db-8a3b-0002a5d5c51b

Defaults

A recorder defaults to the XA_RECORDSTATE_STOPPED state, with no marker, no duration limit, and an update period of 1000 milliseconds, there are no markers set nor callbacks registered and the callback event flags are cleared.

Callbacks

xaRecordCallback

```
typedef void (XAAPIENTRY * xaRecordCallback) (
    XARecordItf caller,
    void * pContext,
    XAuint32 event
);
```

Description	Notifies the recorder application of a recording event.		
Parameters	caller	[in]	Interface on which this callback was registered.
	pContext	[in]	User context data that is supplied when the callback method is registered.
	event	[in]	Event that has occurred (see XA_RECORDEVENT macro).
Comments	None		
See Also	RegisterCallback()		

Methods

SetRecordState			
XAresult (*SetRecordState) (**XARecordItf self,** **XAuint32 state** **);**			
Description	Transitions recorder into the given record state.		
Pre-conditions	None. The recorder may be in any state.		
Parameters	`self`	[in]	Interface self-reference.
	`state`	[in]	Desired recorder state.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	All state transitions are legal.		

GetRecordState			
XAresult (*GetRecordState) (**XARecordItf self,** **XAuint32 * pState** **);**			
Description	Gets the recorder's current record state.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pState`	[out]	Pointer to a location to receive the current record state of the recorder. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetDurationLimit			
XAresult (*SetDurationLimit) (**XARecordItf self,** **XAmillisecond msec** **);**			
Description	Sets the duration of current content in milliseconds.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`msec`	[in]	Non-zero limit on the duration of total recorded content in milliseconds.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	When the recorder reaches the limit, it automatically transitions to the `XARECORDSTATE_STOPPED` state and notifies the application via the `XARECORDEVENT_HEADATLIMIT` event.		

GetPosition			
XAresult (*GetPosition) (**XARecordItf self,** **XAmillisecond * pMsec** **);**			
Description	Returns the current position of the recording head relative to the beginning of content.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMsec`	[out]	Pointer to a location to receive the position of the recording head relative to the beginning of the content, expressed in milliseconds. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The position is synonymous with the amount of recorded content.		

RegisterCallback			
XAresult (*RegisterCallback) (**XARecordItf self,** **xaRecordCallback callback,** **void * pContext** **);**			
Description	Registers the record callback function.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Callback function invoked when one of the specified events occurs. A `NULL` value indicates that there is no callback.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetCallbackEventsMask			
XAresult (*SetCallbackEventsMask) (**XARecordItf self,** **XAuint32 eventFlags** **);**			
Description	Sets the notification state of record events.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`eventFlags`	[in]	Combination record event flags indicating which callback events are enabled. See `XA_RECORDEVENT` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The callback event flags default to all flags cleared.		

GetCallbackEventsMask

```
XAresult (*GetCallbackEventsMask) (
    XARecordItf self,
    XAuint32 * pEventFlags
);
```

Description	Queries the notification state of record events.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pEventFlags`	[out]	Pointer to a location to receive the combination of record event flags indicating which callback events are enabled. This must be non-`NULL`. See `XA_RECORDEVENT` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetMarkerPosition

```
XAresult (*SetMarkerPosition) (
    XARecordItf self,
    XAmillisecond mSec
);
```

Description	Sets the position of the recording marker.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`mSec`	[in]	Position of the marker expressed in milliseconds and relative to the beginning of the content. Must be between 0 and the specified duration limit.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The player will notify the application when the recording head passes through the marker via a callback with a `XARECORDEVENT_HEADATMARKER` event.		

ClearMarkerPosition

```
XAresult (*ClearMarkerPosition) (
    XARecordItf self
);
```

Description	Clears marker.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	This function succeeds even if the marker is already clear.		
See Also	`SetMarkerPosition()`		

GetMarkerPosition

```
XAresult (*GetMarkerPosition) (
    XARecordItf self,
    XAmillisecond * pMSec
);
```

Description	Queries the position of the recording marker.		
Pre-conditions	A marker has been set (using `SetMarkerPosition()` with no intervening `ClearMarkerPosition()`).		
Parameters	`self`	[in]	Interface self-reference.
	`pMSec`	[out]	Pointer to a location to receive the position of the marker expressed in milliseconds and relative to the beginning of the content. Must be between 0 and the specified duration limit. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

SetPositionUpdatePeriod			
XAresult (*SetPositionUpdatePeriod) (**XARecordItf self,** **XAmillisecond mSec** **);**			
Description	Sets the interval between periodic position notifications.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`mSec`	[in]	Non-zero period between position notifications in milliseconds.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The recorder will notify the application when the recording head passes through the positions implied by the specified period. Those positions are defined as the whole multiples of the period relative to the beginning of the content.		

GetPositionUpdatePeriod			
XAresult (*GetPositionUpdatePeriod) (**XARecordItf self,** **XAmillisecond * pMSec** **);**			
Description	Queries the interval between periodic position notifications.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMSec`	[out]	Pointer to a location to receive the period between position notifications in milliseconds. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

8.33 XASeekItf

Description

XASeekItf is an interface for manipulating a playback head, including setting its position and looping characteristics. When supported, seeking may be used, regardless of playback state or rate.

This interface is an implicit interface of Media Player objects (see section 7.4).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_SEEK;

struct XASeekItf_;
typedef const struct XASeekItf_ * const * XASeekItf;

struct XASeekItf_ {
    XAresult (*SetPosition) (
        XASeekItf self,
        XAmillisecond pos,
        XAuint32 seekMode
    );
    XAresult (*SetLoop) (
        XASeekItf self,
        XAboolean loopEnable,
        XAmillisecond startPos,
        XAmillisecond endPos
    );
    XAresult (*GetLoop) (
        XASeekItf self,
        XAboolean * pLoopEnabled,
        XAmillisecond * pStartPos,
        XAmillisecond * pEndPos
    );
};
```

Interface ID

ee6a3120-f776-11db-b518-0002a5d5c51b

Defaults

The playback position defaults to 0 milliseconds (the beginning of the current content). Global and local looping are disabled by default.

Methods

SetPosition			
XAresult (*SetPosition) (**XASeekItf self,** **XAmillisecond pos,** **XAuint32 seekMode** **);**			
Description	Sets the position of the playback head.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pos`	[in]	Desired playback position in milliseconds, relative to the beginning of content.
	`seekMode`	[in]	Inherent seek mode. See the seek mode definition (see section 9.2.71) for details. If the seek mode is not supported, this method will return `XA_RESULT_FEATURE_UNSUPPORTED`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The implementation may set the position to the nearest discrete sample or frame. Note that the position is defined relative to the content playing at 1x forward rate; positions do not scale with changes in playback rate.		

SetLoop			
XAresult (*SetLoop) (**XASeekItf self,** **XAboolean loopEnable,** **XAmillisecond startPos,** **XAmillisecond endPos** **);**			
Description	Enables or disables looping and sets the start and end points of looping. When looping is enabled and the playback head reaches the end position, the player automatically sets the head to the start position and remains in the XA_PLAYSTATE_PLAYING state. Setting a loop does not otherwise have any effect on the playback head even if the head is outside the loop at the time the loop is set.		
Pre-conditions	Specified end position is greater than specified start position.		
Parameters	self	[in]	Interface self-reference.
	loopEnable	[in]	Specifies whether looping is enabled (true) or disabled (false).
	startPos	[in]	Position in milliseconds relative to the beginning of content specifying the start of the loop.
	endPos	[in]	Position in milliseconds relative to the beginning of content specifying the end the loop. endPos must be greater than startPos. A value of XA_TIME_UNKNOWN denotes the end of the stream.
Return value	The return value can be one of the following: XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID		
Comments	If local looping is not supported, this method returns XA_RESULT_FEATURE_UNSUPPORTED.		

GetLoop			
XAresult (*GetLoop) (**XASeekItf self,** **XAboolean * pLoopEnabled,** **XAmillisecond * pStartPos,** **XAmillisecond * pEndPos** **);**			
Description	Queries whether looping is enabled or disabled, and retrieves loop points.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pLoopEnabled`	[out]	Pointer to a location to receive the flag indicating whether looping is enabled (true) or disabled (false). This must be non-`NULL`.
	`pStartPos`	[out]	Pointer to a location to receive the position in milliseconds relative to the beginning of content specifying the start of the loop. This must be non-`NULL`.
	`pEndPos`	[out]	Pointer to a location to receive the postion in milliseconds relative to the beginning of content specifying the end the loop. A value of `XA_TIME_UNKNOWN` denotes the end of the stream. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		

8.34 XASnapshotItf

Description

This interface is for controlling the photographing of still images with a camera device. It contains the `InitiateSnapshot()` method for prefetching the shooting for minimizing the delay with the actual taking of the photo which is done by the `TakeSnapshot()` method. There are also mechanisms for controlling the shutter feedback (sound) and for querying for burst shooting capabilities.

Snapshots can be stored to a file system specified by the `XADataSink` parameter of the `InitiateSnapshot()` method, or if the `XADataSink` specified is `NULL`, the application gets the image data directly as a memory buffer via the `xaSnapshotTakenCallback()` callback. This direct passing of the memory buffer is not supported with burst shooting. Please see the documentation of the methods below for details.

This is a mandated interface of Media Recorder objects (see section 7.5). See section D.5 for an example using this interface.

Call Sequence

The following is the typical call sequence for taking a photo.

1. Set-up:

- `XAImageEncoderItf::SetImageSettings()` (for choosing the codec and the resolution)
- `XACameraItf::SetAutoLocks(self, 0)` (free all locks)
- `XACameraItf` (use various methods to set up flash, zoom, exposure, focus, white balance etc.)

2. Memory allocation:

- `XASnapshotItf::InitiateSnapshot()`
  - ➔ `xaSnapshotInitiatedCallback`

3. Halfway press (for locking the automatic settings that can be locked pre-exposure)**:**

- `XACameraItf::SetAutoLocks(self, XA_CAMERA_AUTO_LOCK_FOCUS | XA_CAMERA_AUTO_LOCK_EXPOSURE)` (an example to lock auto focus and auto exposure)
  - ➔ `xaCameraCallback(context, XA_CAMERACBEVENT_FOCUSSTATUS, XA_CAMERA_FOCUSMODESTATUS_REACHED)`
  - ➔ `xaCameraCallback(context, XA_CAMERACBEVENT_EXPOSURESTATUS, XA_CAMERA_AUTOEXPOSURESTATUS_SUCCESS)`

4. Full press:

- `XASnapshotItf::TakeSnapshot()`

  ➔ `xaSnapshotTakenCallback`

Prototype

```
XA_API extern const XAInterfaceID XA_IID_SNAPSHOT;

struct XASnapshotItf_;
typedef const struct XASnapshotItf_ * const * XASnapshotItf;

struct XASnapshotItf_ {
    XAresult (*InitiateSnapshot) (
        XASnapshotItf self,
        XAuint32 numberOfPictures,
        XAuint32 fps,
        XAboolean freezeViewFinder,
        XADataSink * pSink,
        xaSnapshotInitiatedCallback initiatedCallback,
        xaSnapshotTakenCallback takenCallback,
        void * pContext
    );
    XAresult (*TakeSnapshot) (
        XASnapshotItf self
    );
    XAresult (*CancelSnapshot) (
        XASnapshotItf self
    );
    XAresult (*ReleaseBuffers) (
        XASnapshotItf self,
        const XADataSink * pImage
    );
    XAresult (*GetMaxPicsPerBurst) (
        XASnapshotItf self,
        XAuint32 * pMaxNumberOfPictures
    );
    XAresult (*GetBurstFPSRange) (
        XASnapshotItf self,
        XAuint32 * pMinFPS,
        XAuint32 * pMaxFPS
    );
    XAresult (*SetShutterFeedback) (
        XASnapshotItf self,
        XAboolean enabled
    );
    XAresult (*GetShutterFeedback) (
        XASnapshotItf self,
        XAboolean * pEnabled
    );
};
```

Interface ID

e9d1c860-d71c-11df-8953-0002a5d5c51b

Defaults

No callback registered.

Callbacks

xaSnapshotInitiatedCallback			

`typedef void (XAAPIENTRY * xaSnapshotInitiatedCallback) (`
`  XASnapshotItf caller,`
`  void * context`
`);`			

Description	This method is called when the snapshot shooting has been initiated.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`context`	[in]	User context data that is supplied when the callback method is registered.
Comments	None		
See also	None		

xaSnapshotTakenCallback			
typedef void (XAAPIENTRY * xaSnapshotTakenCallback) (XASnapshotItf caller, void * context, XAuint32 numberOfPicsTaken, const XADataSink * pImage);			
Description	This method is called when the snapshot has been taken.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`context`	[in]	Callback context passed to `XASnapshotItf::InitiateSnapshot()`
	`numberOfPicsTaken`	[in]	The number of snapshots taken if the shooting was successful; zero if snapshots couldn't be taken because some reason, including out of memory situations and situations when forced flash wasn't loaded.
	`pImage`	[in]	A memory address data sink the picture that was taken. The application should use `ReleaseBuffer()` to free the allocated memory after the application has completed processing with the image data. Please note that if a data sink was specified with `InitiateSnapshot()` method, the picture(s) is/are stored in the location defined by the `XADataSink` instead.
Comments	A hint: remember to unfreeze the viewfinder by changing the viewfinder player to the playing state after some time if the viewfinder was frozen. Or, you might want to ask the end user if he or she wants to save the image before unfreezing. In some implementations, the saving (to the `XADataSink` specified by `InitiateSnapshot()` method) can be cancelled by calling `CancelSnapshot()` method while the viewfinder is frozen; if `CancelSnapshot()` is not called, the picture will by stored automatically once the viewfinder is unfrozen. See `CancelSnapshot()` method for details. Implementations may chose not to support a NULL data sink specified in the `InitiateSnapshot()` method and return XA_FEATURE_UNSUPPORTED.		
See also	None		

Methods

InitiateSnapshot			
XAresult (*InitiateSnapshot) (**XASnapshotItf self,** **XAuint32 numberOfPictures,** **XAuint32 fps,** **XAboolean freezeViewFinder,** **const XADataSink * pSink,** **xaSnapshotInitiatedCallback initiatedCallback,** **xaSnapshotTakenCallback takenCallback,** **void * pContext** **);**			
Description	This method prepares the device for snapshot to shorten the actual shooting delay with `TakeSnapshot()` method. The various settings for snapshot are set with this method: the specified number of snapshots, the output location either to the `XADataSink` (if it is specified) or to memory (if no `XADataSink` is specified) and then calls `xaSnapshotInitiatedCallback()` method. Asynchronous. Second call of this method before the call to `TakeSnapshot()` method will reinitialize the shooting with the new parameter values.		
Pre-conditions	The image settings need to be set with `XAImageEncoderItf::SetImageSettings()` method prior calling this method (unless the default image settings will be used).		
Parameters	`self`	[in]	Interface self-reference.
	`numberOfPictures`	[in]	Number of pictures that will be taken. If the number is larger than one, camera will take snapshots in a burst mode consequently as fast as it cans. Zero value here cancels the shooting. This cannot be smaller than zero or larger than `GetMaxPicsPerBurst()`.
	`fps`	[in]	Hint to the device that how many pictures per second should be taken in burst mode. This parameter is ignored if `numberOfPictures` is one.
	`freezeViewfinder`	[in]	If true, freezes the viewfinder (for preview) once the picture has been taken by changing the viewfinder player's state to paused. If multiple pictures are about to be taken only the last picture shot will be frozen on the viewfinder. If false, does not freeze the viewfinder once the picture has been taken. Please note that in some implementations it is still possible to cancel the saving of the picture once the viewfinder is frozen by calling `CancelSnapshot()` method. If `CancelSnapshot()` has not been called once the viewfinder is unfrozen again, the picture will be then stored. See `CancelSnapshot()` for details.

InitiateSnapshot			
	pSink	[in]	XADataSink where to store the resulting images. If sink is NULL, the resulting image will be stored instead into memory (reserved by the implementation) and the address of that memory buffer will be given with the xaSnapshotTakenCallback() in a new implementation generated data sink of the type XA_DATALOCATOR_ADDRESS. If this option is used, numberOfPicture parameter must be 1. The XADataSink should be of type XA_DATALOCATOR_URI and the URI should specify (besides of protocol and (optional) path) the filename prefix. The structure of the generated file name is <prefix><number><file_extension>. prefix is a user given prefix taken from the URI. number is a string generated automatically by the implementation. The generated string is a zero padded four digit running number starting from 0. Therefore, the sequence of generated strings is "0000", "0001", "0002", etc. The length of the string increases if more digits are needed to represent the number so "9999" will be followed by "10000". If a file with the same name exists in the file system (directory) already the next number will be used. file_extension is a system generated extension to the file. It could be for example ".jpg." The system should use the extension corresponding to the chosen encoding.
	initiatedCallback	[in]	Address of the callback.
	takenCallback	[in]	Address of the callback.
	pContext	[in]	User context data that is to be returned as part of the callback method.
Return value	The return value can be one of the following: XA_RESULT_SUCCESS XA_RESULT_PARAMETER_INVALID		
Comments	None		
See also	XADataSink documentation for details, for example, on specifying a file system directory using an URL. TakeSnapshot(). Implementations may chose not to support a NULL sink parameter. Such implementations would return XA_FEATURE_UNSUPPORTED.if this method is called with a NULL sink.		

TakeSnapshot			
XAresult (*TakeSnapshot) (XASnapshotItf self);			
Description	This method takes the specified number of snapshots, stores them either to the `XADataSink` (if it is specified by `InitiateSnapshot()`) or to memory (if no `XADataSink` is specified) and then calls `xaSnapshotTakenCallback()` method. Asynchronous.		
Pre-conditions	`xaSnapshotInitiatedCallback()` must have been called before this method is called.		
Parameters	`self`	[in]	Interface self-reference.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	None		
See also	`InitiateSnapshot(), CancelSnapshot()`		

CancelSnapshot			
XAresult (*CancelSnapshot) (XASnapshotItf self);			
Description	This method cancels an ongoing shooting session. Snapshooting needs to be initiated again after calling this method with InitiateSnapshot method. Synchronous.		
Pre-conditions	Shooting session must be going on.		
Parameters	`self`	[in]	Interface self-reference.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	Please note that in some implementations it is still possible to cancel the saving of the picture once the viewfinder is frozen (for preview) by calling this `CancelSnapshot()` method. Some implementations save the picture directly to `XADataSink` already immediately once it is shot; in that case `CancelSnapshot()` returns `XA_RESULT_PRECONDITIONS_VIOLATED` during freezing (previewing).		
See also	None		

ReleaseBuffer			
XAresult (*ReleaseBuffers) (**XASnapshotItf self,** **const XADataSink * pImage** **);**			
Description	This method releases the given buffer.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pImage`	[in]	Memory address data sink to be released.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`xaSnapshotTakenCallback()`		

GetMaxPicsPerBurst			
XAresult (*GetMaxPicsPerBurst) (**XASnapshotItf self,** **XAuint32 * pMaxNumberOfPictures** **);**			
Description	This method tells how many pictures it is possible to be taken during single burst.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMaxNumberOfPictures`	[in]	Maximum number of pictures that the device supports to be taken in a single burst.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

GetBurstFPSRange			
XAresult (*GetBurstFPSRange) (**XASnapshotItf self,** **XAuint32 * pMinFPS,** **XAuint32 * pMaxFPS** **);**			
Description	This method tells the range of shooting rates possible in burst shooting mode. Please note that these rates might be different depending on which encoder and which resolution has been chosen; not all the rates can necessarily be reached with every resolution or encoder.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMinFPS`	[out]	Minimum rate supported in frames per second.
	`pMaxFPS`	[out]	Maximum rate supported in frames per second. This is zero if burst mode is not supported by the device.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

SetShutterFeedback			
XAresult (*SetShutterFeedback) (**XASnapshotItf self,** **XAboolean enabled** **);**			
Description	Toggles the shutter feedback (such as shutter sound or some visual feedback while taking a snapshot).		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`enabled`	[in]	True to enable shutter feedback; false to disable it.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	Some implementations will return `XA_RESULT_FEATURE_UNSUPPORTED` when trying to switch off the feedback in case shutter feedback is mandatory because of legislative reasons.		
See also	None		

GetShutterFeedback			
XAresult (*GetShutterFeedback) (**XASnapshotItf self,** **XAboolean* pEnabled** **);**			
Description	This method tells if the shutter feedback (such as shutter sound or some visual feedback while taking a snapshot) is enabled.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pEnabled`	[out]	True if shutter feedback is enabled; false if it is disabled.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

8.35 XAStreamInformationItf

Description

`XAStreamInformationItf` is used to query a stream's properties.

This interface is a mandatory interface of Media Player (see section 7.4) and Metadata Extractor objects (see section 7.6

Prototype

```
XA_API extern const XAInterfaceID XA_IID_STREAMINFORMATION;

struct XAStreamInformationItf_;
typedef const struct XAStreamInformationItf_ * const *
XAStreamInformationItf;

struct XAStreamInformationItf_ {
    XAresult (*QueryMediaContainerInformation) (
        XAStreamInformationItf self,
        XAMediaContainerInformation * pInfo
    );
    XAresult (*QueryStreamType) (
        XAStreamInformationItf self,
        XAuint32 streamIndex,
        XAuint32 * pDomain
    );
    XAresult (*QueryStreamInformation) (
        XAStreamInformationItf self,
        XAuint32 streamIndex,
        void * pInfo
    );
    XAresult (*QueryStreamName) (
        XAStreamInformationItf self,
        XAuint32 streamIndex,
        XAuint16 * pNameSize,
        XAchar * pName
    );
    XAresult (*RegisterStreamChangeCallback) (
        XAStreamInformationItf self,
        xaStreamEventChangeCallback callback,
        void * pContext
    );
   XAresult (*QueryActiveStreams) (
        XAStreamInformationItf self,
        XAuint32 * pNumStreams,
        XAboolean * pActiveStreams
    );
```

```
    XAresult (*SetActiveStream) (
         XAStreamInformationItf self,
         XAuint32   streamNum,
         XAboolean  active,
         XAboolean  commitNow
     );
};
```

Interface ID

75e0e2a0-d71d-11df-8c92-0002a5d5c51b

Callbacks

xaStreamEventChangeCallback

```
typedef void (XAAPIENTRY * xaStreamEventChangeCallback) (
    XAStreamInformationItf caller,
    XAuint32 eventId,
    XAuint32 streamIndex,
    void * pEventData,
    void * pContext
);
```

Description	Executes whenever a stream event has changed. Upon this notification, the application may query for the new stream information via `QueryStreamInformation()`.		
Parameters	`caller`	[in]	Interface on which this callback was registered.
	`eventID`	[in]	Identifies the type of notification callback being report. Refer to `XA_STREAMCBEVENT` for a list of available events.
	`streamIndex`	[in]	Identifies the stream with the property change.
	`pEventData`	[in]	Specifies additional information specific to a notification callback event. The contents of this parameter is depedent on the event being reported.
	`pContext`	[in]	User context data that is supplied when the callback method is registered.
Comments	When the streamIndex parameter returns the reserved value of 0 (Media Container Identification) it indicates that a change in the number of available streams has been detected within the media container. `QueryMediaContainerInformation()` shall be used to determine the new number of available streams within the media container.		
See Also	`RegisterStreamChangeCallback()`, `QueryStreamInformation()`,`QueryMediaContainerInformation()`		

Methods

QueryMediaContainerInformation			
XAresult (*QueryMediaContainerInformation) (**XAStreamInformationItf self,** **XAMediaContainerInformation * pInfo** **);**			
Description	Queries information about the media container.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`pInfo`	[out]	Structure containing the media container information.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTENT_CORRUPTED`
	`XA_RESULT_CONTENT_UNSUPPORTED`		

Comments	None		

QueryStreamType			
XAresult (*QueryStreamType) (**XAStreamInformationItf self,** **XAuint32 streamIndex,** **XAuint32 * pDomain** **);**			
Description	Queries the individual streams to determine which domain they are based with.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`streamIndex`	[in]	Incrementing index used to query the available streams. Supported index range is 1 to N, where N is the number of streams available. The value 0 is a reserved value that shall always represent the Media Container.
	`pDomain`	[out]	Identifies the stream domain.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTENT_CORRUPTED`
	`XA_RESULT_CONTENT_UNSUPPORTED`		

Comments	None		
See Also	QueryMediaContainerInformation()		

QueryStreamInformation			
XAresult (*QueryStreamInformation) (XAStreamInformationItf self, XAuint32 streamIndex, void * pInfo);			
Description	Queries information about the stream.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`streamIndex`	[in]	Index identifying the stream within the container that is being queried. The stream index value is same stream index indentifer that is obtained via QueryStreamType
	`pInfo`	[out]	Structure containing the stream information. The structure type definition is associated as per the domain parameter setting – refer to Table 14.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTENT_CORRUPTED`
	`XA_RESULT_CONTENT_UNSUPPORTED`
	
	It is not possible to query Information for streams identified as `XA_DOMAINTYPE_UNKNOWN`, any attempt to do so shall return a result of `XA_RESULT_CONTENT_UNSUPPORTED`.		

Comments	None		

Table 14 : Stream Information Structures vs Domain Types

Value	Associated Structure
XA_DOMAINTYPE_AUDIO	XAAudioStreamInformation
XA_DOMAINTYPE_VIDEO	XAVideoStreamInformation
XA_DOMAINTYPE_IMAGE	XAImageStreamInformation
XA_DOMAINTYPE_TIMEDTEXT	XATimedTextStreamInformation
XA_DOMAINTYPE_VENDOR	XAVendorStreamInformation
XA_DOMAINTYPE_MIDI	XAMIDIStreamInformation
XA_DOMAINTYPE_UNKNOWN	Unknown type

QueryStreamName

```
XAresult (*QueryStreamName) (
    XAStreamInformationItf self,
    XAuint32 streamIndex,
    XAuint16 * pNameSize,
    XAchar * pName
);
```

Description	Queries information about the media container.		
Pre-conditions	None.		
Parameters	`self`	[in]	Interface self-reference.
	`streamIndex`	[in]	Index identifying the stream within the container that is being queried. The stream index value is same stream index indentifer that is obtained via QueryStreamType
	`pNameSize`	[in/out]	This is used both as input and output. On input it bounds the size of the stream's string name buffer. On output it specifies the size of the stream's string name. Returns 0 is a stream name is not available.
	`pName`	[out]	This is a string buffer containing the name of the stream. If this pointer is NULL, the pNameSize parameter identifies the length of the stream's name. This allows the application to properly size the buffer that will contain the stream name. The character coding is UTF-8.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTENT_CORRUPTED`
	`XA_RESULT_CONTENT_UNSUPPORTED`		

Comments	For streams identified as XA_DOMAINTYPE_UNKNOWN, it may still be possible to retrieve the stream name (if any exists). If the method is not able to retrieve the stream name, it will identify this by returning a value of 0 via the pNameSize parameter (refer to the pNameSize parameter for more information). Obtaining the stream specific names may be useful for content that contain multiple alternative tracks. For example, DVD content can contain multiple audio language tracks, the application may utilize the stream names - if present – to enumerate and populate the selection list for the user.		

RegisterStreamChangeCallback			
XAresult (*RegisterStreamChangeCallback) (**XAStreamInformationItf self,** **xaStreamEventChangeCallback callback,** **void * pContext** **);**			
Description	Sets the callback for stream property change event notifications.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`callback`	[in]	Specifies the callback method.
	`pContext`	[in]	User context data that is to be returned as part of the callback method.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See Also	`xaStreamEventChangeCallback()`		

QueryActiveStreams			
XAresult (*QueryActiveStreams) (**XAStreamInformationItf self,** **XAuint32 * pNumStreams,** **XAboolean * pActiveStreams** **);**			
Description	Returns the active state for all streams.		
Pre-conditions	Must not be called on a Metadata Extractor object.		
Parameters	`self`	[in]	Interface self-reference.
	`pNumStreams`	[in/out]	Size of the 'pActiveStreams' array. If 'pActiveStreams' is null, QueryActiveStreams will fill in the desired size of the array. This will be equal to the total number of streams.
	`pActiveStreams`	[out]	An array of XAboolean values indicating which streams are active.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_BUFFER_INSUFFICIENT`		

Comments	If 'numStreams' is less than the total number of streams, the implementation may elect to return `XA_RESULT_BUFFER_INSUFFICIENT`. This function shall not be called on a metadata extractor object, and shall return `XA_RESULT_PRECONDITIONS_VIOLATED` in this case.		

SetActiveStream			
XAresult (*SetActiveStream) (XAStreamInformationItf self, XAuint32 streamNum, XAboolean active, XAboolean commitNow);			
Description	Set/unset the active state for a specified stream. The commitNow parameter allows a number of changes to be deferred and then committed at once.		
Pre-conditions	Must not be called on a Metadata Extractor object.		
Parameters	`self`	[in]	Interface self-reference.
	`streamNum`	[in]	The stream on which to set the active state.
	`active`	[in]	Active state to set on the stream.
	`commitNow`	[in]	Perform change immediately.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	It is implementation specific if multiple active streams per-domain type are allowed. No changes will be made to the active state of any stream until this function is called with a commitNow value of `XA_BOOLEAN_TRUE`. The implementation may return `XA_RESULT_FEATURE_UNSUPPORTED` if it does not support selecting multiple active streams, if the specified stream type is unsupported, or if the current set of selected streams cannot be made active. This function shall not be called on a metadata extractor object, and shall return `XA_RESULT_PRECONDITIONS_VIOLATED` in this case.		

8.36 XAThreadSyncItf

Description

Registered callbacks can be invoked concurrently to application threads and even concurrently to other callbacks. The application cannot assume anything about the pContext from which registered callbacks are invoked, and thus using the native synchronization mechanisms for synchronization of callback contexts is not guaranteed to work.

For this purpose, a critical section mechanism is introduced. There is one critical section per engine object. Applications that require more flexibility can implement such a mechanism on top of this critical section mechanism.

The semantics of the critical section mechanism are specified as follows:

- The engine is said to be **in a critical section state** during the time between when a call to `EnterCriticalSection()` has returned successfully and until the time when a call to `ExitCriticalSection()` is made.
- When the engine is in a critical section state, any call to `EnterCriticalSection()` will block until the engine exited the critical section state, or until an error has occurred (the return code of the `EnterCriticalSection()` call will reflect which of the conditions has occurred).

One important point is worth mentioning: when the engine is operating in non-thread-safe mode, the `EnterCriticalSection()` and `ExitCriticalSection()` methods are **not thread safe**, in the sense that their behavior is undefined, should the application call them from within multiple **applicaton contexts** concurrently. These methods will, however, work properly when invoked from a single application context in concurrency with one or more **callback contexts**.

This interface is supported on the engine object (see section 7.2).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_THREADSYNC;

struct XAThreadSyncItf_;
typedef const struct XAThreadSyncItf_ * const * XAThreadSyncItf;

struct XAThreadSyncItf_ {
    XAresult (*EnterCriticalSection) (
        XAThreadSyncItf self
    );
    XAresult (*ExitCriticalSection) (
        XAThreadSyncItf self
    );
};
```

Interface ID

f3599ea0-f776-11db-b3ea-0002a5d5c51b

Defaults

Not in critical section state.

Methods

EnterCriticalSection			

`XAresult (*EnterCriticalSection) (`
`  XAThreadSyncItf self`
`);`			

Description	Blocks until the engine is not in critical section state, then transitions the engine into critical section state.		
Pre-conditions	The calling context must not already be in critical section state.		
Parameters	`self`	[in]	Synchronization interface.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	Use this method to achieve synchronization between application context and callback context(s), or between multiple callback contexts. See comments in the description section regarding thread-safety of this method.		
See also	`ExitCriticalSection()`		

ExitCriticalSection			

`XAresult (*ExitCriticalSection) (`
`  XAThreadSyncItf self`
`);`			

Description	Transitions the engine from critical section state to non-critical section state.		
Pre-conditions	The engine must be in critical section state. The call must be made from the same context that entered the critical section.		
Parameters	`self`	[in]	Synchronization interface.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	Use this method to achieve synchronization between client application context and callback context(s), or between multiple callback contexts. See comment in description section regarding thread-safety of this method.		
See also	`EnterCriticalSection()`		

8.37 XAVibraItf

Description

`XAVibraItf` interface is used to activate and deactivate the Vibra I/O device object, as well as to set its frequency and intensity, if supported.

`XAVibraItf` uses the following state model, which indicates whether the vibration device is vibrating or not:

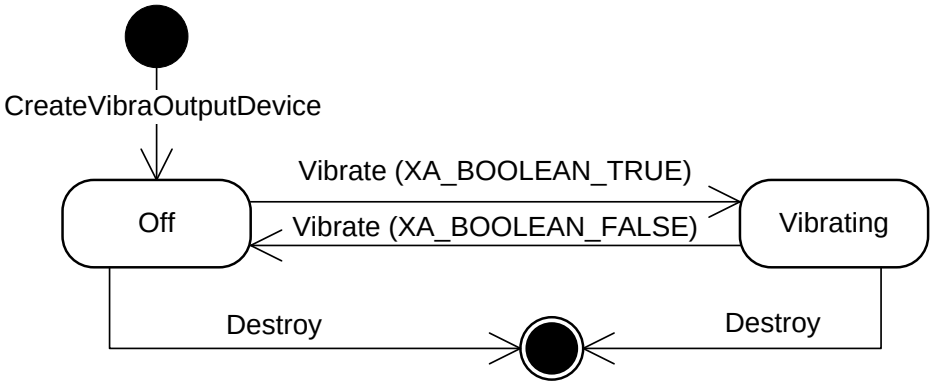


Figure 20: Vibra I/O device state model

This interface is supported on the Vibra I/O device object (see section 7.9).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_VIBRA;

struct XAVibraItf_;
typedef const struct XAVibraItf_ * const * XAVibraItf;

struct XAVibraItf_ {
    XAresult (*Vibrate) (
        XAVibraItf self,
        XAboolean vibrate
    );
    XAresult (*IsVibrating) (
        XAVibraItf self,
        XAboolean * pVibrating
    );
    XAresult (*SetFrequency) (
        XAVibraItf self,
        XAmilliHertz frequency
    );
    XAresult (*GetFrequency) (
        XAVibraItf self,
        XAmilliHertz * pFrequency
    );
```

```
    XAresult (*SetIntensity) (
        XAVibraItf self,
        XApermille intensity
    );
    XAresult (*GetIntensity) (
        XAVibraItf self,
        XApermille * pIntensity
    );
};
```

Interface ID

fe374c00-f776-11db-a8f0-0002a5d5c51b

Defaults

Initially, the object is in the off state. Default frequency and intensity are undefined.

Methods

Vibrate			
`XAresult (*Vibrate) ( XAVibraItf self, XAboolean vibrate );`			
Description	Activates or deactivates vibration for the I/O device.		
Pre-conditions	None.		
Parameters	`self`	[in]	Pointer to a `XAVibraItf` interface.
	`vibrate`	[in]	Boolean indicating whether to vibrate.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_IO_ERROR`
	`XA_RESULT_CONTROL_LOST`		

Comments	None.		
See also	None.		

IsVibrating			
XAresult (*IsVibrating) (**XAVibraItf self,** **XAboolean * pVibrating** **);**			
Description	Returns whether the I/O device is vibrating.		
Pre-conditions	None.		
Parameters	`self`	[in]	Pointer to a `XAVibraItf` interface.
	`pVibrating`	[out]	Address to store a Boolean indicating whether the I/O device is vibrating.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	None.		

SetFrequency			
XAresult (*SetFrequency) (**XAVibraItf self,** **XAmilliHertz frequency** **);**			
Description	Sets the vibration frequency of the I/O device.		
Pre-conditions	The Vibra I/O device must support setting intensity, per `XAVibraDescriptor::supportsFrequency`.		
Parameters	`self`	[in]	Pointer to a `XAVibraItf` interface.
	`frequency`	[in]	Frequency of vibration. Range is [`XAVibraDescriptor::minFrequency`, `XAVibraDescriptor::maxFrequency`]

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_RESOURCE_LOST`
	`XA_RESULT_CONTROL_LOST`		

Comments	None.		
See also	None.		

GetFrequency			
XAresult (*GetFrequency) (**XAVibraItf self,** **XAmilliHertz * pFrequency** **);**			
Description	Returns the vibration frequency of the I/O device.		
Pre-conditions	The Vibra I/O device must support setting intensity, per `XAVibraDescriptor::supportsFrequency`.		
Parameters	`self`	[in]	Pointer to a `XAVibraItf` interface.
	`pFrequency`	[out]	Address to store the vibration frequency. Range is [`XAVibraDescriptor::minFrequency`, `XAVibraDescriptor::maxFrequency`]

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	None.		

SetIntensity			
XAresult (*SetIntensity) (**XAVibraItf self,** **XApermille intensity** **);**			
Description	Sets the vibration intensity of the Vibra I/O device.		
Pre-conditions	The Vibra I/O device must support setting intensity, per `XAVibraDescriptor::supportsIntensity`.		
Parameters	`self`	[in]	Pointer to a `XAVibraItf` interface.
	`intensity`	[in]	Intensity of vibration. Range is [0, 1000].

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_CONTROL_LOST`		

Comments	None.		
See also	None.		

GetIntensity			
XAresult (*GetIntensity) (**const XAVibraItf self,** **XApermille * pIntensity** **);**			
Description	Returns the vibration intensity of the Vibra I/O device.		
Pre-conditions	The Vibra I/O device must support setting intensity, per `XAVibraDescriptor::supportsIntensity`.		
Parameters	`self`	[in]	Pointer to a `XAVibraItf` interface.
	`pIntensity`	[out]	Address to store the vibration intensity of the Vibra I/O device. Range is [0, 1000].

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PRECONDITIONS_VIOLATED`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	None.		

8.38 XAVideoDecoderCapabilitiesItf

Description

This interface provides methods of querying the video decoding capabilities of the media engine.

This interface provides a means of enumerating all video decoders available on an engine where each an decoderId represents each decoder. It also provides a means to query the capabilities of each decoder. A given decoder may support several profile/level pairs each with their own capabilities (such as maximum resolution) appropriate to that profile and level pair. Therefore, this interface represents the capabilities of a particular decoder as a list of capability entries queriable by decoderID and capability entry index.

The set of video decoders supported by the engine does not change during the lifetime of the engine though dynamic resource constraints may limit actual availability when an video decoder is requested.

This interface is a mandated interface of engine objects (see section 7.2).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_VIDEODECODERCAPABILITIES;

struct XAVideoDecoderCapabilitiesItf_;
typedef const struct XAVideoDecoderCapabilitiesItf_
    * const * XAVideoDecoderCapabilitiesItf;

struct XAVideoDecoderCapabilitiesItf_ {
    XAresult (*GetVideoDecoders) (
        XAVideoDecoderCapabilitiesItf self,
        XAuint32 * pNumDecoders,
        XAuint32 * pDecoderIds
    );
    XAresult (*GetVideoDecoderCapabilities) (
        XAVideoDecoderCapabilitiesItf self,
        XAuint32 decoderId,
        XAuint32 * pIndex,
        XAVideoCodecDescriptor * pDescriptor
    );
};
```

Interface ID

d18cb200-e616-11dc-ab01-0002a5d5c51b

Defaults

Not applicable.

Methods

GetVideoDecoders			
XAresult (*GetVideoDecoders) (**XAVideoDecoderCapabilitiesItf self,** **XAuint32 * pNumDecoders,** **XAuint32 * pDecoderIds** **);**			
Description	Retrieves available video decoders.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pNumDecoders`	[in/out]	If `pDecoderIds` is `NULL`, `pNumDecoders` returns the number of decoders available. All implementations must have at least one decoder. If `pDecodersIds` is non-`NULL`, as an input `pNumDecoders` specifies the size of the `pDecoderIds` array and as an output it specifies the number of decoder IDs available within the `pDecoderIds` array.
	`pDecoderIds`	[out]	Array of video decoders provided by the engine. Refer to `XA_VIDEOCODEC` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`GetVideoDecoderCapabilities()`		

GetVideoDecoderCapabilities			
XAresult (*GetVideoDecoderCapabilities) (**XAVideoDecoderCapabilitiesItf self,** **XAuint32 decoderId,** **XAuint32 * pIndex,** **XAVideoCodecDescriptor * pDescriptor** **);**			
Description	Retrieves video decoder capabilities.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`decoderId`	[in]	Specifies video decoder. Refer to `XA_VIDEOCODEC` macros.
	`pIndex`	[in/out]	If `pDescriptor` is `NULL`, pIndex returns the number of video decoders capability descriptions. Each decoder must support at least one profile/mode pair and therefore have at least one Codec Descriptor. If `pDescriptor` is non-`NULL`, pIndex is a incrementing value used to enumerate capability descriptions. Supported index range is 0 to N-1, where N is the number of video decoders capability descriptions.
	`pDescriptor`	[out]	Structure defining the capabilities of the video decoder.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`GetVideoDecoders()`		

8.39 XAVideoEncoderItf

Description

This interface is used to set the parameters to be used by an video encoder.

This interface is a mandated interface of Media Recorder objects (see section 7.5).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_VIDEOENCODER;

struct XAVideoEncoderItf_;
typedef const struct XAVideoEncoderItf_ * const * XAVideoEncoderItf;

struct XAVideoEncoderItf_ {
    XAresult (*SetVideoSettings) (
        XAVideoEncoderItf self,
        XAVideoSettings * pSettings
    );
    XAresult (*GetVideoSettings) (
        XAVideoEncoderItf self,
        XAVideoSettings * pSettings
    );
};
```

Interface ID

9444db60-df06-11db-b311-0002a5d5c51b

Defaults

No default settings are mandated.

Methods

SetVideoSettings			
XAresult (*SetVideoSettings) (XAVideoEncoderItf self, XAVideoSettings * pSettings);			
Description	Set video encoder settings.		
Pre-conditions	RecordItf shall be in stopped state.		
Parameters	`self`	[in]	Interface self-reference.
	`pSettings`	[in]	Video encoder settings. `XA_RESULT_FEATURE_UNSUPPORTED` is returned if the requested encoder is not supported.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_PRECONDITIONS_VIOLATED`		

Comments	None		
See also	`GetImageSetting()`		

GetVideoSettings			
XAresult (*GetVideoSettings) (XAVideoEncoderItf self, XAVideoSettings * pSettings);			
Description	Get video encoder settings.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pSettings`	[out]	Video encoder settings.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The getter returns the exact value that was set by the previous call to the setter. That is, the `keyFrameInterval` field of the returned `XAVideoSettings` struct is zero if the frequency of keyframes is set to be determined automatically.		
See also	`SetImageSetting()`		

8.40 XAVideoEncoderCapabilitiesItf

Description

This interface provides methods of querying the video decoding capabilities of the media engine.

This interface provides a means of enumerating all video encoders available on an engine where each an encoderId represents each encoder. It also provides a means to query the capabilities of each encoder. A given encoder may support several profile/level pairs each with their own capabilities (such as maximum resolution) appropriate to that profile and level pair. Therefore, this interface represents the capabilities of a particular encoder as a list of capability entries queriable by encoderID and capability entry index.

The set of video encoders supported by the engine does not change during the lifetime of the engine though dynamic resource constraints may limit actual availability when an video encoder is requested.

This interface is a mandated interface of engine objects (see section 7.2).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_VIDEOENCODERCAPABILITIES;

struct XAVideoEncoderCapabilitiesItf_;
typedef const struct XAVideoEncoderCapabilitiesItf_
    * const * XAVideoEncoderCapabilitiesItf;

struct XAVideoEncoderCapabilitiesItf_ {
    XAresult (*GetVideoEncoders) (
        XAVideoEncoderCapabilitiesItf self,
        XAuint32 * pNumEncoders,
        XAuint32 * pEncoderIds
    );
    XAresult (*GetVideoEncoderCapabilities) (
        XAVideoEncoderCapabilitiesItf self,
        XAuint32 encoderId,
        XAuint32 * pIndex,
        XAVideoCodecDescriptor * pDescriptor
    );
};
```

Interface ID

5aef2760-e872-11db-849f-0002a5d5c51b

Defaults

Not applicable.

Methods

<table>
<tr><th colspan="4">GetVideoEncoders</th></tr>
<tr><td colspan="4"><pre>XAresult (*GetVideoEncoders) (
    XAVideoEncoderCapabilitiesItf self,
    XAuint32 * pNumEncoders,
    XAuint32 * pEncoderIds
);</pre></td></tr>
<tr><td>Description</td><td colspan="3">Retrieves available video encoders.</td></tr>
<tr><td>Pre-conditions</td><td colspan="3">None</td></tr>
<tr><td rowspan="3">Parameters</td><td>self</td><td>[in]</td><td>Interface self-reference.</td></tr>
<tr><td>pNumEncoders</td><td>[in/out]</td><td>If pEncoderIds is NULL, pNumEncoders returns the number of encoders available. Returns 0 if there are no encoders.
If pEncodersIds is non-NULL, as an input pNumEncoders specifies the size of the pEncoderIds array and as an output it specifies the number of encoder IDs available within the pEncoderIds array.</td></tr>
<tr><td>pEncoderIds</td><td>[out]</td><td>Array of video encoders provided by the engine. Refer to XA_VIDEOCODEC macros.</td></tr>
<tr><td>Return value</td><td colspan="3">The return value can be one of the following:
XA_RESULT_SUCCESS
XA_RESULT_PARAMETER_INVALID</td></tr>
<tr><td>Comments</td><td colspan="3">PROFILE NOTES
A Media Player/Recorder profile implementation must support at least one encoder.</td></tr>
<tr><td>See also</td><td colspan="3">GetVideoEncoderCapabilities()</td></tr>
</table>

GetVideoEncoderCapabilities			
XAresult (*GetVideoEncoderCapabilities) (**XAVideoEncoderCapabilitiesItf self,** **XAuint32 encoderId,** **XAuint32 * pIndex,** **XAVideoCodecDescriptor * pDescriptor** **);**			
Description	Retrieves video encoder capabilities.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`encoderId`	[in]	Specifies video encoder. Refer to `XA_VIDEOCODEC` macros.
	`pIndex`	[in/out]	If pCapabilities is `NULL`, `pIndex` returns the number of capabilities. Each encoder must support at least one profile/mode pair and therefore have at least one Codec Descriptor. If pCapabilities is non-`NULL`, pIndex is an incrementing value used for enumerating profiles. Supported index range is 0 to N-1, where N is the number of capabilities of the encoder.
	`pDescriptor`	[out]	Structure defining the capabilities of the video encoder.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`GetVideoEncoders()`		

8.41 XAVideoPostProcessingItf

Description

The video post-processing interface provides operations on video data. It is realized on an object that supports image or video content. Post-processing operations are carried out in the following order:

1. Cropping
2. Rotating
3. Mirroring
4. Scaling

All the changes will only be committed once `Commit()` is called. This allows clients to apply a group of changes to the video stream in one shot.

How to position the image data on the output screen is out of scope of this interface.

Scaling

Scaling is defined in this API by using three methods: `SetSourceRectangle`, `SetDestinationRectangle` and `SetScaleOptions`. `SetSourceRectangle` defines the rectangle in the original frame that is to be used for further processing. `SetDestinationRectangle` defines then the size of the processed output frame. The amount of scaling applied is then determined by the size difference between the source rectangle and destination rectangle together with the scaleOptions parameter of the SetScaleOptions method. Figure 21 shows examples of scaling between source and destination rectangles of different sizes by using each of the three scale options (STRETCH, FIT and CROP).

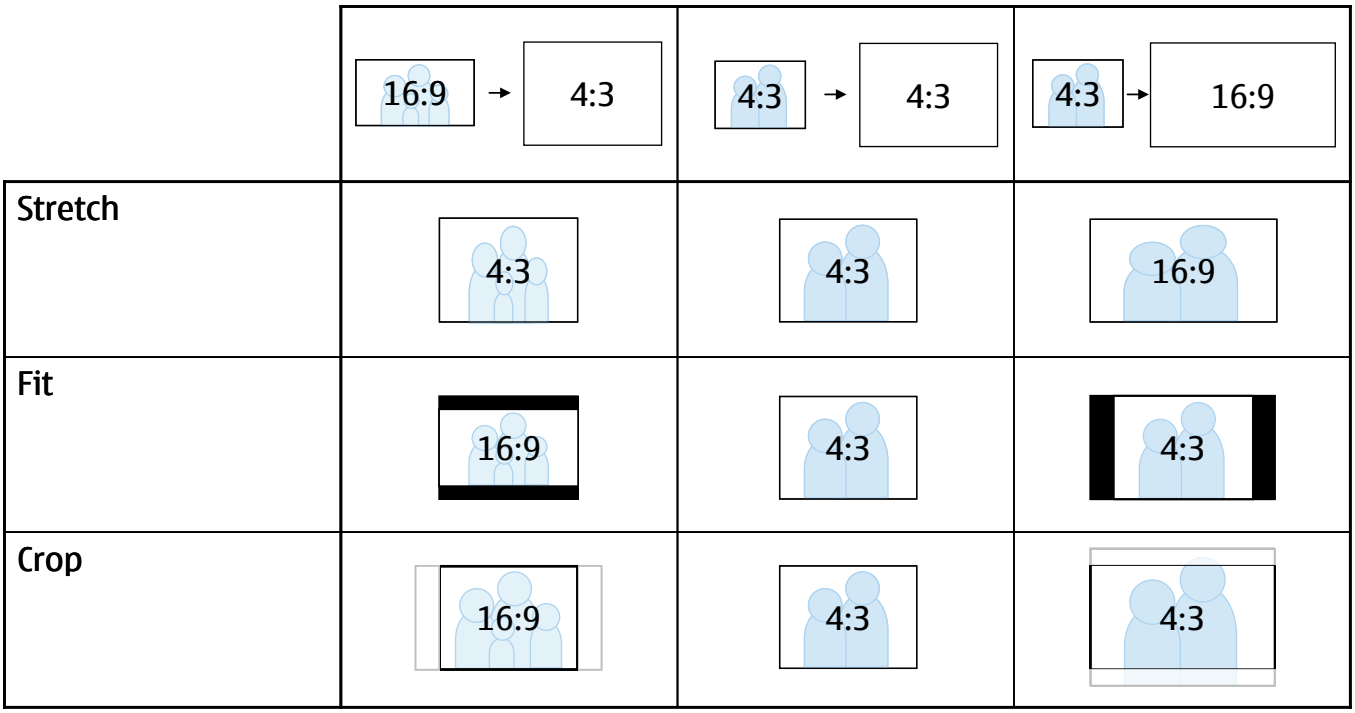


Figure 21. Scaling examples using the three scale options.

Figure 22 is an example of cropping and scaling. Figure 22 (a) is the original video frame with aspect ratio 4:3. Figure 22 (b) is the cropped video with aspect ratio 1:1. It would be the result of calling `SetSourceRectangle` with a 1:1 rectangle (i.e. a square) as the input parameter. Figure 22 (c) is the scaled frame with the scale option 'Fit' to the destination rectangle which has the aspect ratio 16:9. It would be the result of calling SetScaleOptions with `XA_VIDEOSCALE_FIT`, 0 (for black) and `XA_RENDERINGHINT_NONE` as input parameters, followed by `SetDestinationRectangle` with a 16:9 rectangle as the input parameter.

(a) **(b)** **(c)**

Figure 22. Example of cropping and scaling

Prototype

```
XA_API extern const XAInterfaceID XA_IID_VIDEOPOSTPROCESSING;

struct XAVideoPostProcessingItf_;
typedef const struct XAVideoPostProcessingItf_ * const *
XAVideoPostProcessingItf;

struct XAVideoPostProcessingItf_ {
    XAresult (*SetRotation) (
        XAVideoPostProcessingItf self,
        XAmillidegree rotation
    );
    XAresult (*IsArbitraryRotationSupported) (
        XAVideoPostProcessingItf self,
        XAboolean * pSupported
    );
    XAresult (*SetScaleOptions) (
        XAVideoPostProcessingItf self,
        XAuint32 scaleOptions,
        XAuint32 backgroundColor,
        XAuint32 renderingHints
    );
    XAresult (*SetSourceRectangle) (
        XAVideoPostProcessingItf self,
        const XARectangle * pSrcRect
    );
    XAresult (*SetDestinationRectangle) (
        XAVideoPostProcessingItf self,
        const XARectangle * pDestRect
    );
    XAresult (*SetMirror) (
        XAVideoPostProcessingItf self,
        XAuint32 mirror
    );
    XAresult (*Commit) (
        XAVideoPostProcessingItf self
    );
};
```

Interface ID

898b6820-7e6e-11dd-8caf-0002a5d5c51b

Defaults

The default behavior is no rotation, no scaling, no cropping and no mirroring. Therefore, both the source and destination rectangles are, by default, of the original frame size. The default scale options are:

- scaleOptions: `XA_VIDEOSCALE_FIT`
- backgroundColor: 0 (black and not transparent)

- renderingHints: `XA_RENDERINGHINT_NONE` (no hint)

Methods

SetRotation()			
XAresult (*SetRotation) (XAVideoPostProcessingItf self, XAmillidegree rotation);			
Description	Sets post-processing options for rotation.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`rotation`	[in]	Defines the clock-wise rotation angle.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The change will only be committed once `Commit()` is called. Not all implementations will be able to support arbitrary rotation angles. If `IsArbitraryRotationSupported()` tells false and the application tries an angle other than an integer multiple of 90 degrees, `XA_RESULT_FEATURE_UNSUPPORTED` will be returned. All implementations are mandated to support angles that are integer multiples of 90 degrees. Those angles include, but are not limited to 0, 90000, 180000 and 270000 millidegrees.		
See also	`IsArbitraryRotationSupported()`		

IsArbitraryRotationSupported()

```
XAresult (*IsArbitraryRotationSupported) (
    XAVideoPostProcessingItf self,
    XAboolean * pSupported
);
```

Description	Determines if arbitrary rotation angles are supported by the implementation.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pSupported`	[out]	Is set to `XA_BOOLEAN_TRUE` if arbitrary rotation angles are supported, set to `XA_BOOLEAN_FALSE` otherwise.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`SetRotation()`		

SetSourceRectangle()

```
XAresult (*SetSourceRectangle) (
    XAVideoPostProcessingItf self,
    const XARectangle * pSrcRect
);
```

Description	Defines the rectangle in the original frame that is to be used for further processing.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pSrcRect`	[in]	Define the source rectangle to use.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`		

Comments	The change will only be committed once Commit() is called.		
See also	`SetDestinationRectangle()`		

SetDestinationRectangle()			
XAresult (*SetDestinationRectangle) (XAVideoPostProcessingItf self, const XARectangle * pDestRect);			
Description	Defines the destination rectangle for the processed frame. This rectangle, in conjunction with the scaling options used (fit, crop, stretch) determines the scaling applied to the frame.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pDestRect`	[in]	Define the destination rectangle to use.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`
	`XA_RESULT_MEMORY_FAILURE`		

Comments	The change will only be committed once Commit() is called. The method works with SetSourceRectangle() to do cropping and scaling. If SetSourceRectangle() is not called, by default, the original frame size will be used as the source rectangle.		
See also	`SetSourceRectangle() and SetScaleOptions().`		

SetScaleOptions ()			
XAresult (*SetScaleOptions) (XAVideoPostProcessingItf self, XAuint32 scaleOptions XAuint32 backgroundColor, XAuint32 renderingHints);			
Description	Sets the options for scaling.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`scaleOptions`	[in]	Defines the scale option. There are three options: `XA_VIDEOSCALE_STRETCH` - The source and destination rectangle's width and height parameters are used to calculate the scaling factors independently. Aspect ratio is ignored. `XA_VIDEOSCALE_FIT` - The minimum scale factor between the destination rectangle's width over the source rectangle's width and the destination rectangle's height over the source rectangle's height is used. Aspect ratio is maintained. Frame is centered. `XA_VIDEOSCALE_CROP` - The maximum scale factor between the destination rectangle's width over the source rectangle's width and the destination rectangle's height over the source rectangle's height is used. Aspect ratio is maintained. Frame is centered. Figure 21 illustrates these three options.
	`backgroundColor`	[in]	32-bit RGBA color value with 8 bits each for red, green, blue and alpha. This color will be used to fill the borders when scaleOptions is set to `XA_VIDEOSCALE_FIT`. As an example, the cell in Figure 21 'Fit' row and 16:9->4:3 column has black borders on its top and bottom parts. The alpha value specifies the transparency level of those borders only and does not affect the alpha value of the actual video frame.
	`renderingHints`	[in]	Defines the rendering hints to use during scaling. Refer to `XA_RENDERINGHINT` macros.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	`SetDestinationRectangle()`.		

SetMirror			
XAresult (*SetMirror) (**XAVideoPostProcessingItf self,** **XAuint32 mirror** **);**			
Description	Sets post-processing options for mirroring.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`mirror`	[in]	Defines the mirroring type for video post-processors. `XA_RESULT_PARAMETER_INVALID` is returned if an unsupported mirroring type is requested. Refer to XA_VIDEOMIRROR macro.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The change will only be committed once Commit() is called.		
See also	`XA_VIDEOMIRROR.`		

Commit			
XAresult (*Commit) (**XAVideoPostProcessingItf self** **);**			
Description	Commit all video post-processing changes since the last `Commit()`.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
Return value	The return value can be one of the following: `XA_RESULT_SUCCESS`		
Comments	All the changes will only be committed once `Commit()` is called. This ensures that (a) all the changes are applied to the same video frame, and (b) there are no intermediate or orphan frames created during this post-processing.		
See also	`None.`		

8.42 XAVolumeItf

Description

This interface exposes controls for manipulating the object's audio volume properties.

This interface additionally exposes a stereo position control. Its exact effect is determined by the object's format; if the object's format is mono, a pan effect is applied, and if the object's format is stereo, a balance effect is applied.

This interface is supported on the Media Player (see section 7.4) and Output Mix objects (see section 7.7), and may be optionally supported on other objects such as the Media Recorder object (see section 7.5).

Prototype

```
XA_API extern const XAInterfaceID XA_IID_VOLUME;

struct XAVolumeItf_;
typedef const struct XAVolumeItf_ * const * XAVolumeItf;

struct XAVolumeItf_ {
    XAresult (*SetVolumeLevel) (
        XAVolumeItf self,
        XAmillibel level
    );
    XAresult (*GetVolumeLevel) (
        XAVolumeItf self,
        XAmillibel * pLevel
    );
    XAresult (*GetMaxVolumeLevel) (
        XAVolumeItf self,
        XAmillibel * pMaxLevel
    );
    XAresult (*SetMute) (
        XAVolumeItf self,
        XAboolean mute
    );
    XAresult (*GetMute) (
        XAVolumeItf self,
        XAboolean * pMute
    );
    XAresult (*EnableStereoPosition) (
        XAVolumeItf self,
        XAboolean enable
    );
    XAresult (*IsEnabledStereoPosition) (
        XAVolumeItf self,
        XAboolean * pEnable
    );
    XAresult (*SetStereoPosition) (
        XAVolumeItf self,
        XApermille stereoPosition
    );
```

```
    XAresult (*GetStereoPosition) (
        XAVolumeItf self,
        XApermille * pStereoPosition
    );
};
```

Interface ID

088ba520-f777-11db-a5e3-0002a5d5c51b

Defaults

Volume level: 0 mB

Mute: disabled (not muted)

Stereo position: disabled, 0 ‰ (center)

Methods

SetVolumeLevel			
`XAresult (*SetVolumeLevel) ( XAVolumeItf self, XAmillibel level );`			
Description	Sets the object's volume level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`level`	[in]	Volume level in millibels. The valid range is [`XA_MILLIBEL_MIN`, maximum supported level], where maximum supported level can be queried with the method `GetMaxVolumeLevel()`. The maximum supported level is always at least 0 mB.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	If the object is muted, calls to `SetVolumeLevel()` will still change the internal volume level, but this will have no audible effect until the object is unmuted.		
See also	`SetMute()`		

GetVolumeLevel			
XAresult (*GetVolumeLevel) (**XAVolumeItf self,** **XAmillibel * pLevel** **);**			
Description	Gets the object's volume level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pLevel`	[out]	Pointer to a location to receive the object's volume level in millibels. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

GetMaxVolumeLevel			
XAresult (*GetMaxVolumeLevel) (**XAVolumeItf self,** **XAmillibel * pMaxLevel** **);**			
Description	Gets the maximum supported level.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMaxLevel`	[out]	Pointer to a location to receive the maximum supported volume level in millibels. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The maximum supported level is implementation-dependent, but will always be at least 0 mB.		
See also	None		

SetMute			
XAresult (*SetMute) (**XAVolumeItf self,** **XAboolean mute** **);**			
Description	Mutes or unmutes the object.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`mute`	[in]	If true, the object is muted. If false, the object is unmuted.
Return value	The return value can be the following: `XA_RESULT_SUCCESS`		
Comments	Muting the object does not change the volume level reported by `GetVolumeLevel()`. Calling `SetMute()` with `mute` set to true when the object is already muted is a valid operation that has no effect. Calling `SetMute()` with `mute` set to false when the object is already unmuted is a valid operation that has no effect.		
See also	`GetVolumeLevel()`		

GetMute			
XAresult (*GetMute) (**XAVolumeItf self,** **XAboolean * pMute** **);**			
Description	Retrieves the object's mute state.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pMute`	[out]	Pointer to a Boolean to receive the object's mute state. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

EnableStereoPosition

```
XAresult (*EnableStereoPosition) (
    XAVolumeItf self,
    XAboolean enable
);
```

Description	Enables or disables the stereo positioning effect.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`enable`	[in]	If true, enables the stereo position effect. If false, disables the stereo positioning effect (no attenuation due to stereo positioning is applied to the left or right channels).

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None.		
See also	None.		

IsEnabledStereoPosition

```
XAresult (*IsEnabledStereoPosition) (
    XAVolumeItf self,
    XAboolean * pEnable
);
```

Description	Returns the enabled state of the stereo positioning effect.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pEnable`	[out]	Pointer to a location to receive the enabled state of the stereo positioning effect.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

SetStereoPosition			
XAresult (*SetStereoPosition) (XAVolumeItf self, XApermille stereoPosition);			
Description	Sets the stereo position of the object; For mono objects, this will control a constant energy pan effect, and for stereo objects, this will control a balance effect.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`stereoPosition`	[in]	Stereo position in the range [-1000 ‰, 1000 ‰]. A stereo position of 0 ‰ indicates the object is in the center. That is, in the case of balance, no attenuation is applied to the left and right channels; and in the case of pan, 3 dB attenuation is applied to the left and right channels. A stereo position of –1000 ‰ pans the object fully to the left; the right channel is silent. A stereo position of 1000 ‰ pans the object fully to the right; the left channel is silent.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	The exact pan and balance curves used for this method are implementation-dependent, subject to satisfying the parameter description. For objects whose format is mono, this method controls a constant energy pan effect. For objects whose format is stereo, this method controls a balance effect.		
See also	None		

GetStereoPosition			
XAresult (*GetStereoPosition) (**XAVolumeItf self,** **XApermille * pStereoPosition** **);**			
Description	Gets the object's stereo position setting.		
Pre-conditions	None		
Parameters	`self`	[in]	Interface self-reference.
	`pStereoPosition`	[out]	Pointer to a location to receive the current stereo position setting. This must be non-`NULL`.

Return value	The return value can be one of the following:
	`XA_RESULT_SUCCESS`
	`XA_RESULT_PARAMETER_INVALID`		

Comments	None		
See also	None		

9 Macros and Typedefs

9.1 Structures

9.1.1 XAAudioCodecDescriptor

```
typedef struct XAAudioCodecDescriptor_ {
    XAuint32 maxChannels;
    XAuint32 minBitsPerSample;
    XAuint32 maxBitsPerSample;
    XAmilliHertz minSampleRate;
    XAmilliHertz maxSampleRate;
    XAboolean isFreqRangeContinuous;
    XAmilliHertz * pSampleRatesSupported;
    XAuint32 numSampleRatesSupported;
    XAuint32 minBitRate;
    XAuint32 maxBitRate;
    XAboolean isBitrateRangeContinuous;
    XAuint32 * pBitratesSupported;
    XAuint32 numBitratesSupported;
    XAuint32 profileSetting;
    XAuint32 modeSetting;
    XAuint32 streamFormat;
} XAAudioCodecDescriptor;
```

This structure is used for querying the capabilities of an audio codec.

Field	Description
maxChannels	Maximum number of audio channels.
minBitsPerSample	Minimum bits per sample of PCM data.
maxBitsPerSample	Maximum bits per sample of PCM data.
minSampleRate	Minimum sampling rate supported.
maxSampleRate	Maximum sampling rate supported.
isFreqRangeContinuous	Returns XA_BOOLEAN_TRUE if the device supports a continuous range of sampling rates between minSampleRate and maxSampleRate; otherwise returns XA_BOOLEAN_FALSE.
pSampleRatesSupported	Indexed array containing the supported sampling rates. Ignored if isFreqRangeContinuous is XA_BOOLEAN_TRUE. If pSampleRatesSupported is NULL, the number of supported sample rates is returned in numSampleRatesSupported.
numSampleRatesSupported	Size of the pSamplingRatesSupported array. Ignored if isFreqRangeContinuous is XA_BOOLEAN_TRUE.
minBitRate	Minimum bitrate.
maxBitRate	Maximum bitrate.
isBitrateRangeContinuous	Returns XA_BOOLEAN_TRUE if the device supports a continuous range of bitrates between minBitRate and maxBitRate; otherwise returns XA_BOOLEAN_FALSE.

Field	Description
pBitratesSupported	Indexed array containing the supported bitrates. Ignored if isBitrateRangeContinuous is XA_BOOLEAN_TRUE. If pBitratesSupported is NULL, the number of supported bitrates is returned in numBitratesSupported.
numBitratesSupported	Size of the pBitratesSupported array. Ignored if isBitrateRangeContinuous is XA_BOOLEAN_TRUE.
profileSetting	Profile supported. See XA_AUDIOPROFILE defines [see section 9.2.4].
modeSetting	Level supported. See XA_AUDIOMODE defines [see section 9.2.4].
streamFormat	Stream format of the codec. See XA_AUDIOSTREAMFORMAT defines [see section 9.2.4].

9.1.2 XAAudioEncoderSettings

```
typedef struct XAAudioEncoderSettings_ {
    XAuint32 encoderId;
    XAuint32 channelsIn;
    XAuint32 channelsOut;
    XAmilliHertz sampleRate;
    XAuint32 bitRate;
    XAuint32 bitsPerSample;
    XAuint32 rateControl;
    XAuint32 profileSetting;
    XAuint32 levelSetting;
    XAuint32 channelMode;
    XAuint32 streamFormat;
    XAuint32 encodeOptions;
    XAuint32 blockAlignment;
} XAAudioEncoderSettings;
```

This structure is used to set the audio encoding parameters.

Field	Description
encoderId	Identifies the supported audio encoder. Refer to XA_AUDIOOCODEC macros.
channelsIn	Number of input audio channels.
channelsOut	Number of output channels in encoded data. In case of contradiction between this field and the channelMode field, the channelMode field overrides.
sampleRate	Audio samplerate of input audio data.
bitRate	Bitrate of encoded data.
bitsPerSample	Bits per sample of input data.
rateControl	Encoding rate control mode. See XA_RATECONTROLMODE macros.
profileSetting	Profile to use for encoding. See XA_AUDIOPROFILE macros.
levelSetting	Level to use for encoding. See XA_AUDIOMODE macros.
channelMode	Channel mode for encoder. See XA_AUDIOCHANMODE macros.
streamFormat	Format of encoded bit-stream. For example, AMR encoders use this to select between IF1, IF2, or RTPPAYLOAD bit-stream formats. Refer to XA_AUDIOSTREAMFORMAT_XXX defines in section 9.2.3
encodeOptions	Codec specific encoder options. For example, WMA encoders use it to specify codec version, framesize, frequency extensions, and other options. See the relevant encoder documentatation for format. This is typically a bitfield specifying encode options. . Use a value of zero to specify use of the default encoder settings for the encoder.
blockAlignment	Block alignment in bytes of an audio sample.

9.1.3 XAAudioInputDescriptor

```
typedef struct XAAudioInputDescriptor_ {
    XAchar * pDeviceName;
    XAint16 deviceConnection;
    XAint16 deviceScope;
    XAint16 deviceLocation;
    XAboolean isForTelephony;
    XAmilliHertz minSampleRate;
    XAmilliHertz maxSampleRate;
    XAboolean isFreqRangeContinuous;
    XAmilliHertz * pSamplingRatesSupported;
    XAint16 numOfSamplingRatesSupported;
    XAint16 maxChannels;
} XAAudioInputDescriptor;
```

This structure is used for returning the description of audio input device capabilities. The `deviceConnection`, `deviceScope` and `deviceLocation` fields collectively describe the type of audio input device in a standardized way, while still allowing new device types to be added (by vendor-specific extensions of the corresponding macros), if necessary. For example, on a mobile phone, the integrated microphone would have the following values for each of these three fields, respectively: `XA_DEVCONNECTION_INTEGRATED`, `XA_DEVSCOPE_USER` and `XA_DEVLOCATION_HANDSET`, while a Bluetooth headset microphone would have the following values: `XA_DEVCONNECTION_ATTACHED_WIRELESS`, `XA_DEVSCOPE_USER` and `XA_DEVLOCATION_HEADSET`.

Field	Description
`pDeviceName`	Human-readable string representing the name of the device, such as “Bluetooth microphone” or “wired microphone”.
`deviceConnection`	One of the device connection types listed in the `XA_DEVCONNECTION` macros.
`deviceScope`	One of the device scope types listed in the `XA_DEVSCOPE` macros.
`deviceLocation`	One of the device location types listed in the `XA_DEVLOCATION` macros
`isForTelephony`	Returns `XA_BOOLEAN_TRUE` if the audio input device is deemed suitable for telephony uplink audio; otherwise returns `XA_BOOLEAN_FALSE`. For example: a line-in jack would not be considered suitable for telephony, as it is difficult to determine what can be connected to it.
`minSampleRate`	Minimum sampling rate supported.
`maxSampleRate`	Maximum sampling rate supported.
`isFreqRangeContinuous`	Returns `XA_BOOLEAN_TRUE` if the input device supports a continuous range of sampling rates between `minSampleRate` and `maxSampleRate`; otherwise returns `XA_BOOLEAN_FALSE`.
`pSamplingRatesSupported`	Indexed array containing the supported sampling rates, as defined in the `XA_SAMPLING_RATE` macros. Ignored if `isFreqRangeContinuous` is `XA_BOOLEAN_TRUE`.
`numOfSamplingRatesSupported`	Size of the `samplingRatesSupported` array. Ignored if `isFreqRangeContinuous` is `XA_BOOLEAN_TRUE`.

Field	Description
maxChannels	Maximum number of channels supported; for mono devices, value would be 1.

The table below shows examples of the first five fields of the XAAudioInputDescriptor struct for various audio input devices. For the sake of brevity and clarity, the full names of the XA_DEV macros have been abbreviated to include just the distinct portion of the names (such as XA_DEVCONNECTION_INTEGRATED appears as INTEGRATED and XA_DEVSCOPE_PRIVATE as PRIVATE).

Table 15: Examples of Audio Input Devices.

deviceName	device-Connection	device-Scope	device-Location	isForTelephony
Handset microphone	INTEGRATED	USER	HANDSET	TRUE
Bluetooth microphone	WIRELESS	USER	HEADSET	TRUE
Wired headset microphone	WIRED	USER	HEADSET	TRUE
Carkit microphone	WIRED	ENVIRON-MENT	CARKIT	TRUE
Carkit handset microphone	WIRED	USER	CARKIT	TRUE
System line-in jack	INTEGRATED	UNKNOWN	HANDSET	FALSE
Networked media Server	NETWORK	UNKNOWN	REMOTE	FALSE

9.1.4 XAAudioOutputDescriptor

```
typedef struct XAAudioOutputDescriptor_ {
    XAchar * pDeviceName;
    XAint16 deviceConnection;
    XAint16 deviceScope;
    XAint16 deviceLocation;
    XAboolean isForTelephony;
    XAmilliHertz minSampleRate;
    XAmilliHertz maxSampleRate;
    XAboolean isFreqRangeContinuous;
    XAmilliHertz * pSamplingRatesSupported;
    XAint16 numOfSamplingRatesSupported;
    XAint16 maxChannels;
} XAAudioOutputDescriptor;
```

This structure is used for returning the description of audio output device capabilities. The `deviceConnection`, `deviceScope` and `deviceLocation` fields collectively describe the type of audio output device in a standardized way, while still allowing new device types to be added (by vendor-specific extensions of the corresponding macros), if necessary. For example, on a mobile phone, the earpiece would have the following values for each of these three fields, respectively: `XA_DEVCONNECTION_INTEGRATED`, `XA_DEVSCOPE_USER` and `XA_DEVLOCATION_HANDSET`, while a pair of speakers that are part of a music dock would have the following: `XA_DEVCONNECTION_ATTACHED_WIRED`, `XA_DEVSCOPE_ENVIRONMENT` and `XA_DEVLOCATION_DOCK`.

Field	Description
`pDeviceName`	Human-readable string representing the name of the output device, such as “integrated loudspeaker” or “Bluetooth headset”.
`deviceConnection`	One of the device connection types listed in the `XA_DEVCONNECTION` macros.
`deviceScope`	One of the device scope types listed in the `XA_DEVSCOPE` macros.
`deviceLocation`	One of the device location types listed in the `XA_DEVLOCATION` macros.
`isForTelephony`	Returns `XA_BOOLEAN_TRUE` if the audio output device is deemed suitable for telephony downlink audio; otherwise returns `XA_BOOLEAN_FALSE`. For example, a line-out jack would not be a suitable for telephony downlink audio.
`minSampleRate`	Minimum sampling rate supported.
`maxSampleRate`	Maximum sampling rate supported.
`isFreqRangeContinuous`	Returns `XA_BOOLEAN_TRUE` if the output device supports a continuous range of sampling rates between `minSampleRate` and `maxSampleRate`; otherwise returns `XA_BOOLEAN_FALSE`.
`pSamplingRatesSupported`	Indexed array containing the supported sampling rates, as defined in the `XA_SAMPLINGRATE` macros. Ignored if `isFreqRangeContinuous` is `XA_BOOLEAN_TRUE`.
`numOfSamplingRatesSupported`	Size of the `samplingRatesSupported` array. Ignored if `isFreqRangeContinuous` is `XA_BOOLEAN_TRUE`.

Field	Description
maxChannels	Maximum number of channels supported; for mono devices, value would be 1.

The table below shows examples of the first six fields of the XAAudioOutputDescriptor struct for various audio output devices. For the sake of brevity and clarity, the full names of the XA_DEV macros have been abbreviated to include just the distinct portion of the names (such as XA_DEVCONNECTION_INTEGRATED appears as INTEGRATED and XA_DEVSCOPE_USER as USER).

Table 16: Examples of Audio Output Devices

deviceName	Device-Connection	device-Scope	device-Location	isFor-Telephony
Earpiece	INTEGRATED	USER	HANDSET	TRUE
Loudspeaker	INTEGRATED	ENVIRON-MENT	HANDSET	TRUE
Bluetooth headset speaker	WIRELESS	USER	HEADSET	TRUE
Wired headset speaker	WIRED	USER	HEADSET	TRUE
Carkit loudspeaker	WIRED	ENVIRON-MENT	CARKIT	TRUE
Carkit handset speaker	WIRED	USER	CARKIT	TRUE
System line-out jack	INTEGRATED	UNKNOWN	HANDSET	FALSE
Dock loudspeaker	WIRED	ENVIRON-MENT	DOCK	FALSE
FM radio Transmitter	WIRED	ENVIRON-MENT	DOCK	FALSE
Networked media renderer	NETWORK	UNKNOWN	REMOTE	FALSE

9.1.5 XAAudioStreamInformation

```
typedef struct XAAudioStreamInformation_ {
    XAuint32 codecId;
    XAuint32 channels;
    XAmillHertz sampleRate;
    XAuint32 bitRate;
    XAchar langCountry[16];
    XAmillisecond duration;
} XAAudioStreamInformation;
```

This structure is used for querying the information about an audio stream.

Field	Description
codecId	Identifies the stream's codec format ID (Refer to XA_AUDIOCODEC)
channels	Identifies the number of audio channels within the stream.
sampleRate	Identifies the audio stream's sample rate. If the sample rate is unknown, this value shall be 0.
bitRate	Identies the audio stream's bit rate in units of bits per second. If the bit rate is unknown, this value shall be 0.
langCountry	Language/country code of the stream. (see note below);
duration	Identifies the total duration of the audio stream. If the duration is unknown, this value shall be XA_TIME_UNKNOWN.

The language / country code may be a language code, a language / country code, or a country code.
Formatting of language codes and language / country codes is defined by IETF RFC 3066 [RFC3066] (which incorporates underlying ISO specifications 639 [ISO639] and 3166 [ISO3166] and a syntax). Formatting of country codes is defined by ISO 3166 [ISO3166].

9.1.6 XACameraDescriptor

```
typedef struct XACameraDescriptor_ {
    XAchar * pName;
    XAuint32 maxWidth;
    XAuint32 maxHeight;
    XAuint32 orientation;
    XAuint32 featuresSupported;
    XAuint32 exposureModesSupported;
    XAuint32 flashModesSupported;
    XAuint32 focusModesSupported;
    XAuint32 meteringModesSupported;
    XAuint32 whiteBalanceModesSupported;
} XACameraDescriptor;
```

Structure used to query the camera capabilities. Zoom factor ranges will vary from 1 to the value specified:

Field	Description
pName	Human readable string representing the name of the device.
maxWidth	Maximum supported width in units of pixels.
maxHeight	Maximum supported height in units of pixels.
orientation	Camera mounting orientation. See XA_ORIENTATION macros.
featureSupported	A bitwise OR of camera features supported. See XA_CAMERACAP macros.
exposureModesSupported	A bitwise OR of exposure modes supported. See XA_CAMERA_EXPOSUREMODE macros.
flashModesSupported	A bitwise OR of flash modes supported. See XA_CAMERA_FLASHMODE macros.

Field	Description
focusModesSupported	A bitwise OR of focus modes supported. See XA_CAMERA_FOCUSMODE macros.
meteringModesSupported	A bitwise OR of metering modes supported. See XA_CAMERA_METERINGMODE macros.
whiteBalanceModesSupported	A bitwise OR of white balance modes supported. See XA_CAMERA_WHITEBALANCEMODE macros.

9.1.7 XADataFormat_MIME

```
typedef struct XADataFormat_MIME_ {
    XAuint32 formatType;
    const XAchar * pMimeType;
    XAuint32 containerType;
} XADataFormat_MIME;
```

Fields include:

Field	Description
formatType	The format type, which must always be XA_DATAFORMAT_MIME for this structure.
pMimeType	The mime type of the data expressed as a string.
containerType	The container type of the data. When an application uses this structure to specify the data source for a player use case, the application may leave the containerType unspecified (for example XA_CONTAINERTYPE_UNSPECIFIED) or may provide a specific value as a hint to the player. When an application uses this structure to specify the data sink for a recorder use case, the application is dictating the container type of the captured content.

9.1.8 XADataFormat_PCM

NOTE: This structure is deprecated. Use XADataFormat_PCM_EX instead.

```
typedef struct XADataFormat_PCM_ {
    XAuint32 formatType;
    XAuint32 numChannels;
    XAuint32 samplesPerSec;
    XAuint32 bitsPerSample;
    XAuint32 containerSize;
    XAuint32 channelMask;
    XAuint32 endianness;
} XADataFormat_PCM;
```

Fields include:

Field	Description
formatType	The format type, which must always be XA_DATAFORMAT_PCM for this structure.
numChannels	Numbers of audio channels present in the data. Multi-channel audio is always interleaved in the data buffer.
samplesPerSec	The audio sample rate of the data. **Note: This is in milliHertz and not Hertz, as the field name would suggest.**
bitsPerSample	Number of actual data bits in a sample. If bitsPerSample is equal to 8 then the data's representation is XA_PCM_REPRESENTATION_UNSIGNED_INT. Otherwise, the data's representation is XA_PCM_REPRESENTATION_SIGNED_INT.
containerSize	The container size for PCM data in bits, for example 24 bit data in a 32 bit container. Data is left-justified within the container. For best performance, it is recommended that the container size be the size of the native data types.
channelMask	Channel mask indicating mapping of audio channels to speaker location. The channelMask member specifies which channels are present in the multichannel stream. The least significant bit corresponds to the front left speaker (XA_SPEAKER_FRONT_LEFT), the next least significant bit corresponds to the front right speaker (XA_SPEAKER_FRONT_RIGHT), and so on. The full list of valid speaker locations is defined in section 9.2.73. The channels specified in channelMask must be present in the prescribed order (from least significant bit up). For example, if only XA_SPEAKER_FRONT_LEFT and XA_SPEAKER_FRONT_RIGHT are specified, the samples for the front left speaker must come first in the interleaved stream. The number of bits set in channelMask should be the same as the number of channels specified in numChannels.
endianness	Endianness of the audio data. See XA_BYTEORDER macro for definition.

9.1.9 XADataFormat_PCM_EX

```
typedef struct XADataFormat_PCM_EX__ {
    XAuint32 formatType;
    XAuint32 numChannels;
    XAuint32 sampleRate;
    XAuint32 bitsPerSample;
    XAuint32 containerSize;
    XAuint32 channelMask;
    XAuint32 endianness;
    XAuint32 representataion;
} XADataFormat_PCM_EX;
```

Fields include:

Field	Description
formatType	The format type, which must always be XA_DATAFORMAT_PCM for this structure.
numChannels	Numbers of audio channels present in the data. Multi-channel audio is always interleaved in the data buffer.
sampleRate	The audio sample rate of the data in milliHertz.
bitsPerSample	Number of actual data bits in a sample. If bitsPerSample is equal to 8 then the data's representation is XA_PCM_REPRESENTATION_UNSIGNED_INT. Otherwise, the data's representation is XA_PCM_REPRESENTATION_SIGNED_INT.
containerSize	The container size for PCM data in bits, for example 24 bit data in a 32 bit container. Data is left-justified within the container. For best performance, it is recommended that the container size be the size of the native data types.
channelMask	Channel mask indicating mapping of audio channels to speaker location. The channelMask member specifies which channels are present in the multichannel stream. The least significant bit corresponds to the front left speaker (XA_SPEAKER_FRONT_LEFT), the next least significant bit corresponds to the front right speaker (XA_SPEAKER_FRONT_RIGHT), and so on. The full list of valid speaker locations is defined in section 9.2.73. The channels specified in channelMask must be present in the prescribed order (from least significant bit up). For example, if only XA_SPEAKER_FRONT_LEFT and XA_SPEAKER_FRONT_RIGHT are specified, the samples for the front left speaker must come first in the interleaved stream. The number of bits set in channelMask should be the same as the number of channels specified in numChannels.
endianness	Endianness of the audio data. See XA_BYTEORDER macro for definition.
representataion	Representation of the audio data. See XA_PCM_REPRESENTATION macro for definition.

9.1.10 XADataFormat_RawImage

```
typedef struct XADataFormat_RawImage_ {
    XAuint32 formatType;
    XAuint32 colorFormat;
    XAuint32 height;
    XAuint32 width;
    XAuint32 stride;
} XADataFormat_RawImage;
```

Structure used to describe the raw image data:

Field	Description
formatType	The format type, which must always be XA_DATAFORMAT_RAWIMAGE for this structure.
colorFormat	Raw image color format. Refer to XA_COLORFORMAT macros.
height	Frame height (vertical) resolution.
width	Frame width (horizontal) resolution.
stride	Number of bytes in a line of the image.

9.1.11 XADataLocator_Address

```
typedef struct XADataLocator_Address_ {
    XAuint32 locatorType;
    const void * pAddress;
    XAuint32 length;
} XADataLocator_Address;
```

Fields include:

Field	Description
locatorType	Locator type, which must always be XA_DATALOCATOR_ADDRESS for this structure.
pAddress	Address of the first byte of data.
length	Length of the data in bytes.

9.1.12 XADataLocator_ContentPipe

```
typedef struct XADataLocator_ContentPipe_ {
    XAuint32 locatorType;
    const void * pContentPipe;
    const XAchar * pURI;
} XADataLocator_ContentPipe;
```

Fields include:

Field	Description
locatorType	Locator type, which must be XA_DATALOCATOR_CONTENTPIPE for this structure.
pContentPipe	Pointer to the structure for the content pipe. Refer to the Content Pipe Specification [CP] for details.
pURI	The URI for the content pipe.

9.1.13 XADataLocator_IODevice

```
typedef struct XADataLocator_IODevice_ {
    XAuint32 locatorType;
    XAuint32 deviceType;
    XAuint32 deviceID;
    XAObjectItf device;
} XADataLocator_IODevice;
```

Fields include:

Field	Description
locatorType	Locator type, which must be XA_DATALOCATOR_IODEVICE for this structure.
deviceType	Type of I/O device. See XA_IODEVICE macros.
deviceID	ID of the device. Ignored if device is not NULL.
device	I/O device object itself. Must be NULL if deviceID parameter is to be used.

9.1.14 XADataLocator_MediaObject

```
typedef struct XADataLocator_MediaObject_ {
    XAuint32 locatorType;
    XAObjectItf mediaObject;
} XADataLocator_MediaObject;
```

Fields include:

Field	Description
locatorType	Locator type, which must be XA_DATALOCATOR_MEDIA for this structure.
mediaObject	Media object created by the engine.

9.1.15 XADataLocator_NativeDisplay

```
typedef struct XADataLocator_NativeDisplay_ {
    XAuint32 locatorType;
    XANativeHandle hWindow;
    XANativeHandle hDisplay;
} XADataLocator_NativeDisplay;
```

Fields include:

Field	Description
locatorType	Locator type, which must be XA_DATALOCATOR_NATIVEDISPLAY for this structure.
hWindow	Handle of the native display window.
hDisplay	Handle of the native display.

9.1.16 XADataLocator_Null

```
typedef struct XADataLocator_Null_ {
    XAuint32 locatorType;
} XADataLocator_Null;
```

Fields include:

Field	Description
locatorType	Locator type, which must be XA_DATALOCATOR_NULL for this structure.

9.1.17 XADataLocator_OutputMix

```
typedef struct XADataLocator_OutputMix_ {
    XAuint32 locatorType;
    XAObjectItf outputMix;
} XADataLocator_OutputMix;
```

Fields include:

Field	Description
locatorType	Locator type, which must be XA_DATALOCATOR_OUTPUTMIX for this structure.
outputMix	The OutputMix object as retrieved from the engine.

9.1.18 XADataLocator_URI

```
typedef struct XADataLocator_URI_ {
    XAuint32 locatorType;
    const XAchar * pURI;
} XADataLocator_URI;
```

Fields include:

Field	Description
locatorType	Locator type, which must always be XA_DATALOCATOR_URI for this structure.
pURI	URI expressed as a string.

9.1.19 XADataSink

```
typedef struct XADataSink_ {
    void * pLocator;
    void * pFormat;
} XADataSink;
```

Fields include:

Field	Description
pLocator	Pointer to the specified data locator structure. This may point to any of the following structures. XADataLocator_Address XADataLocator_IODevice XADataLocator_NativeDisplay XADataLocator_OutputMix XADataLocator_URI XADataLocator_MediaObject XADataLocator_Null XADataLocator_ContentPipe The first field of each of these structures includes the 32 bit locatorType field, which identifies the locator type (See XA_DATALOCATOR definitions) and hence the structure pointed to. Note: The available XA_DATALOCATOR definitions may be extended through an API extension.
pFormat	A pointer to the specified format structure. This may point to any of the following structures. XADataFormat_PCM **(Deprecated)** XADataFormat_PCM_EX XADataFormat_MIME XADataFormat_RawImage The first field of each of these structures includes the 32 bit formatType field, which identifies the format type (XA_DATAFORMAT definitions) and hence the structure pointed to. pFormat is ignored if pLocator is XADataLocator_IODevice, XADataLocator_OutputMix, XADataLocator_NativeDisplay, XADataLocator_MediaObject, or XADataLocator_Null.

9.1.20 XADataSource

```
typedef struct XADataSource_ {
    void * pLocator;
    void * pFormat;
} XADataSource;
```

Fields include:

Field	Description
pLocator	Pointer to the specified data locator structure. This may point to any of the following structures. XADataLocator_Address XADataLocator_IODevice XADataLocator_URI XADataLocator_MediaObject XADataLocator_Null XADataLocator_ContentPipe The first field of each of these structures includes the 32 bit locatorType field, which identifies the locator type (see XA_DATALOCATOR definitions) and hence the structure pointed to. Note: The available XA_DATALOCATOR definitions may be extended through an API extension.
pFormat	A pointer to the specified format structure. This may point to any of the following structures. XADataFormat_PCM **(Deprecated)** XADataFormat_PCM_EX XADataFormat_MIME XADataFormat_RawImage The first field of each of these structures includes the 32 bit formatType field, which identifies the format type (XA_DATAFORMAT definitions) and hence the structure pointed to. pFormat is ignored if pLocator is XADataLocator_IODevice.

9.1.21 XAEngineOption

```
typedef struct XAEngineOption_ {
    XAuint32 feature;
    XAuint32 data;
} XAEngineOption;
```

Structure used for specifying different options during engine creation:

Field	Description
feature	Feature identifier. See XA_ENGINEOPTION macros.
data	Value to use for feature.

9.1.22 XAFocusPointPosition

```
typedef struct XAFocusPointPosition_ {
    XAuint32 left;
    XAuint32 top;
    XAuint32 width;
    XAuint32 height;
} XAFocusPointPosition;
```

This structure is used to specify the camera focus region point position and size.

Field	Description
left	The leftmost coordinate of the focus point.
top	The topmost coordinate of the focus point.
width	The width of the focus point.
height	The height of the focus point.

9.1.23 XAHSL

```
typedef struct XAHSL_ {
    XAmillidegree hue;
    XApermille saturation;
    XApermille lightness;
} XAHSL;
```

XAHSL represents a color defined in terms of the HSL color space.

Field	Description
hue	Hue. Range is [0, 360000] in millidegrees. (Refers to the range between 0 and 360 degrees).
saturation	Saturation of the color. Range is [0, 1000] in permille. (Refers to the range between 0.0% and 100.0%).
lightness	Lightness of the color. Range is [0, 1000] in permille. (Refers to the range between 0.0% and 100.0%).

9.1.24 XAImageCodecDescriptor

```
typedef struct XAImageCodecDescriptor_ {
    XAuint32 codecId;
    XAuint32 maxWidth;
    XAuint32 maxHeight;
} XAImageCodecDescriptor;
```

This structure is used to query the capabilities of image encoders and decoders.

Field	Description
codecId	Identifies the supported image codec. Refer to XA_IMAGECODEC macros.
maxWidth	Maximum frame width (horizontal) resolution.
maxHeight	Maximum frame height (vertical) resolution.

9.1.25 XAImageSettings

```
typedef struct XAImageSettings_ {
    XAuint32 encoderId;
    XAuint32 width;
    XAuint32 height;
    XApermille compressionLevel;
    XAuint32 colorFormat;
} XAImageSettings;
```

This structure is used for setting the encoding parameters.

Field	Description
encoderId	Identifies the supported image encoder. Refer to XA_IMAGECODEC macros.
width	Frame width (horizontal) resolution.
height	Frame height (vertical) resolution.
compressionLevel	Compression level is in the range of 0 to 1000. A value of 0 indicates implementation default. A level of 1000 produces the highest compression and a level of 1 produces the lowest compression. Note: This parameter is to be ignored if the encoder does not support this capability.
colorFormat	Color format to use if XA_IMAGECODEC_RAW is specified. Refer to XA_COLORFORMAT macros.

9.1.26 XAImageStreamInformation

```
typedef struct XAImageStreamInformation_ {
    XAuint32 codecId;
    XAuint32 width;
    XAuint32 height;
    XAmillisecond presentationDuration;
} XAImageStreamInformation;
```

This structure is used for querying the information about an image stream.

Field	Description
`codecId`	Identifies the stream's codec format ID (Refer to XA_IMAGECODEC)
`width`	Identifies the image stream's horizontal resolution (width).
`height`	Identifies the image stream's vertical resolution (height).
`presentationDuration`	Identifies the total duration of the image stream. If the duration is unknown, this value shall be XA_TIME_UNKNOWN. For images, this means the total duration that the image should be presented. For image streams that contain multiple images, an event notification will be issued when the next image stream is processed by the object and the presentation duration has changed. This will allow the allocation to query for the updated image presentation duration that is associated with the new image.

9.1.27 XAInterfaceID

```
typedef const struct XAInterfaceID_ {
    XAuint32 time_low;
    XAuint16 time_mid;
    XAuint16 time_hi_and_version;
    XAuint16 clock_seq;
    XAuint8 node[6];
} * XAInterfaceID;
```

The interface ID type.

Field	Description
time_low	Low field of the timestamp.
time_mid	Middle field of the timestamp.
time_hi_and_version	High field of the timestamp multiplexed with the version number.
clock_seq	Clock sequence.
node	Spatially unique node identifier.

9.1.28 XALEDDescriptor

```
typedef struct XALEDDescriptor_ {
    XAuint8 ledCount;
    XAuint8 primaryLED;
    XAuint32 colorMask;
} XALEDDescriptor;
```

XALEDDescriptor represents the capabilities of the LED array I/O Device.

Field	Description
ledCount	Number of LEDs in the array. Range is [1, 32].
primaryLED	Index of the primary LED, which is the main status LED of the device. Range is [0, ledCount-1].
colorMask	Bitmask indicating which LEDs support color. Valid bits range from the least significant bit, which indicates the first LED in the array, to bit ledCount–1, which indicates the last LED in the array.

9.1.29 XAMediaContainerInformation

```
typedef struct XAMediaContainerInformation_ {
    XAuint32 containerType;
    XAmillisecond mediaDuration;
    XAuint32 numStreams;
} XAMediaContainerInformation;
```

XAMediaContainerInformation is used for querying information about a media container.

Field	Description
containerType	Identifies the media container type (Refer to XA_CONTAINERTYPE).
mediaDuration	Identifies the total duration of the content. If the duration is unknown, this value shall be XA_TIME_UNKNOWN.
numStreams	Identifies the number of streams (tracks) available within the media container.

9.1.30 XAMetadataInfo

```
typedef struct XAMetadataInfo_ {
    XAuint32 size;
    XAuint32 encoding;
    XAchar langCountry[16];
    XAuint8 data[1];
} XAMetadataInfo;
```

XAMetadataInfo represents a key or a value from a metadata item key/value pair.

Field	Description
size	Size of the data in bytes. size must be greater than 0.
encoding	Character encoding of the data.
langCountry	Language / country code of the data (see note below).
data	Key or value, as represented by the encoding.

The language / country code may be a language code, a language / country code, or a country code. When specifying the code, note that a partially-specified code will match fully-specified codes that match the part that is specified. For example, “en” will match “en-us” and other “en” variants. Likewise, “us” will match “en-us” and other “us” variants.

Formatting of language codes and language / country codes is defined by IETF RFC 3066 [RFC3066] (which incorporates underlying ISO specifications 639 [ISO639] and 3166 [ISO3166] and a syntax). Formatting of country codes is defined by ISO 3166 [ISO3166].

9.1.31 XAMIDIStreamInformation

```
typedef struct XAMIDIStreamInformation_ {
    XAuint32 channels;
    XAuint32 tracks;
    XAuint32 bankType;
    XAchar langCountry[16];
    XAmillisecond duration;
} XAMIDIStreamInformation;
```

This structure is used for querying the information about a MIDI stream.

Field	Description
channels	Number of MIDI channels. If the number of channels is unknown, the value shall be XA_MIDI_UNKNOWN.
tracks	Number of MIDI tracks. If the number of tracks is unknown, the value shall be XA_MIDI_UNKNOWN.
bankType	Identifies the type of MIDI sound bank(s) used, as defined in XA_MIDIBANK . If the bank information is unknown, the value shall be XA_MIDI_UNKNOWN.
langCountry	Language/country code of the stream. Will be an empty (null-terminated) string in case it is not known or applicable. Also, see note below.
duration	Identifies the total duration of the MIDI stream. If the duration is unknown, this value shall be XA_TIME_UNKNOWN.

The language / country code may be a language code, a language / country code, or a country code.
Formatting of language codes and language / country codes is defined by IETF RFC 3066 [RFC3066] (which incorporates underlying ISO specifications 639 [ISO639] and 3166 [ISO3166] and a syntax). Formatting of country codes is defined by ISO 3166 [ISO3166].

9.1.32 XANativeHandle

```
typedef void * XANativeHandle;
```

An opaque handle native to the platform that represents a display or window.

9.1.33 XARectangle

```
typedef struct XARectangle_ {
    XAuint32 left;
    XAuint32 top;
    XAuint32 width;
    XAuint32 height;
} XARectangle;
```

This structure is used to specify a rectangle.

Field	Description
left	Horizontal position of the leftmost column of pixels.
top	Vertical position of the topmost row of pixels.
width	The rectangle's width in pixels.
height	The rectangle's height in pixels.

9.1.34 XATimedTextStreamInformation

```
typedef struct XATimedTextStreamInformation_ {
    XAuint16 layer;
    XAuint32 width;
    XAuint32 height;
    XAuint16 tx;
    XAuint16 ty;
    XAuint32 bitrate;
    XAchar langCountry[16];
    XAmillisecond duration;
} XATimedTextStreamInformation;
```

This structure is used for querying the information about a timed text stream. These values represent the default settings described within the stream and may be overwritten with updated settings within the stream as the stream is processed.

Field	Description
layer	Specifies the layering depth (similar to z-index in SMIL) of the text in relation to other rendered video content from this container. A greater negative value represents a distance closer to the viewer.
width	Specifies the width of the text region in the original video coordinates.
height	Specifies the height of the text region in the original video coordinates.
tx	Specifies the X position value for the text region in relation to the original video area.
ty	Specifies the Y position value for the text region in relation to the original video area.
bitrate	Specifies the bitrate of the stream, in units of bits per second.
langCountry	Language/country code of the stream. (see note below);
duration	Identifies the total duration of the audio stream. If the duration is unknown, this value shall be XA_TIME_UNKNOWN.

The language / country code may be a language code, a language / country code, or a country code.

Formatting of language codes and language / country codes is defined by IETF RFC 3066 [RFC3066] (which incorporates underlying ISO specifications 639 [ISO639] and 3166 [ISO3166] and a syntax). Formatting of country codes is defined by ISO 3166 [ISO3166].

9.1.35 XAVendorStreamInformation

```
typedef struct XAVendorStreamInformation_ {
    void *VendorStreamInfo;
    } XAVendorStreamInformation;
```

This structure is used for querying the information about a vendor-specific stream.

Field	Description
VendorStreamInfo	Information about this vendor-specific stream

9.1.36 XAVibraDescriptor

```
typedef struct XAVibraDescriptor_ {
    XAboolean supportsFrequency;
    XAboolean supportsIntensity;
    XAmilliHertz minFrequency;
    XAmilliHertz maxFrequency;
} XAVibraDescriptor;
```

`XAVibraDescriptor` represents the capabilities of the Vibra I/O device.

Field	Description
`supportsFrequency`	Boolean indicating whether the Vibra I/O device supports setting the frequency of vibration.
`supportsIntensity`	Boolean indicating whether the Vibra I/O device supports setting the intensity of vibration.
`minFrequency`	Minimum frequency supported by the Vibra I/O device. Range is [1, `XA_MILLIHERTZ_MAX`]. If `supportsFrequency` is set to `XA_BOOLEAN_FALSE`, this will be set to 0.
`maxFrequency`	Maximum frequency supported by the Vibra I/O device. Range is [`minFrequency`, max XAmilliHertz]. If `supportsFrequency` is set to `XA_BOOLEAN_FALSE`, this will be set to 0.

9.1.37 XAVideoCodecDescriptor

```
typedef struct XAVideoCodecDescriptor_ {
    XAuint32 codecId;
    XAuint32 maxWidth;
    XAuint32 maxHeight;
    XAuint32 maxFrameRate;
    XAuint32 maxBitRate;
    XAuint32 rateControlSupported;
    XAuint32 profileSetting;
    XAuint32 levelSetting;
} XAVideoCodecDescriptor;
```

This structure is used to query the capabilities of video encoders and decoders.

Field	Description
`codecId`	Identifies the supported video codec. Refer to `XA_VIDEOCODEC` macros.
`maxWidth`	Maximum frame width (horizontal) resolution.
`maxHeight`	Maximum frame height (vertical) resolution.
`maxFrameRate`	Maximum encoding frame rate in units of frames per second. This value is represented in Q16 format, where the upper 16 bits represent the integer value and the lower 16 bits represent the fractional value.
`maxBitRate`	Maximum encoding bitrate in units of bits per second.

Field	Description
rateControlSupported	Encoding rate control modes supported. See XA_RATECONTROLMODE macros. This field is only valid for encoders, otherwise it is reserved.
profileSetting	Profile supported by codec. See XA_VIDEOPROFILE macros.
levelSetting	Level supported by codec. See XA_VIDEOLEVEL macros.

9.1.38 XAVideoSettings

```
typedef struct XAVideoSettings_ {
    XAuint32 encoderId;
    XAuint32 width;
    XAuint32 height;
    XAuint32 frameRate;
    XAuint32 bitRate;
    XAuint32 rateControl;
    XAuint32 profileSetting;
    XAuint32 levelSetting;
    XAuint32 keyFrameInterval;
} XAVideoSettings;
```

This structure is used to set the video encoding parameters.

Field	Description
encoderId	Identifies the supported video encoder. Refer to XA_VIDEOCODEC macros.
width	Frame width (horizontal) resolution.
height	Frame height (vertical) resolution.
frameRate	Encoding frame rate in units of frames per second. This value is represented in Q16 format, where the upper 16 bits represent the integer value and the lower 16 bits represent the fractional value.
bitRate	Encoding bitrate in units of bits per second.
rateControl	Encoding rate control mode. See XA_RATECONTROLMODE macros.
profileSetting	Profile to use for encoding. See XA_VIDEOPROFILE macros.
levelSetting	Level to use for encoding. See XA_VIDEOLEVEL macros.
keyFrameInterval	Number of frames between keyframes. A value of 0 indicates that frequency of keyframes to be determined automatically.

9.1.39 XAVideoStreamInformation

```
typedef struct XAVideoStreamInformation_ {
    XAuint32 codecId;
    XAuint32 width;
    XAuint32 height;
    XAuint32 frameRate;
    XAuint32 bitRate;
    XAmillisecond duration;
} XAVideoStreamInformation;
```

This structure is used for querying the information about a video stream.

Field	Description
codecId	Identifies the stream's codec format ID (Refer to XA_VIDEOCODEC)
width	Identifies the video stream's horizontal resolution (width).
height	Identifies the video stream's vertical resolution (height).
frameRate	Identifies the video stream's frame rate in units of frames per second. If the frame rate is unknown, this value shall be 0. This value is represented in Q16 format, where the upper 16 bits represent the integer value and the lower 16 bits represent the fractional value.
bitRate	Identifies the video stream's bit rate in units of bits per second. If the bit rate is unknown, this value shall be 0.
duration	Identifies the total duration of the video stream. If the duration is unknown, this value shall be XA_TIME_UNKNOWN.

9.2 Macros

9.2.1 XA_API

```
#define XA_API <system dependent>
```

A platform-specific macro used to declare OPENMAX AL function prototypes. It is modified to meet the requirements for a particular platform

9.2.2 XAAPIENTRY

```
#define XAAPIENTRY <system dependent>
```

A system-dependent API entry point macro. This may be used to indicate the required calling conventions for global functions.

9.2.3 XA_AUDIOCODEC

```
#define XA_AUDIOCODEC_PCM        ((XAuint32) 0x00000001)
#define XA_AUDIOCODEC_MP3        ((XAuint32) 0x00000002)
#define XA_AUDIOCODEC_AMR        ((XAuint32) 0x00000003)
#define XA_AUDIOCODEC_AMRWB      ((XAuint32) 0x00000004)
#define XA_AUDIOCODEC_AMRWBPLUS  ((XAuint32) 0x00000005)
#define XA_AUDIOCODEC_AAC        ((XAuint32) 0x00000006)
#define XA_AUDIOCODEC_WMA        ((XAuint32) 0x00000007)
#define XA_AUDIOCODEC_REAL       ((XAuint32) 0x00000008)
#define XA_AUDIOCODEC_VORBIS     ((XAuint32) 0x00000009)
```

These macros are used for setting the audio encoding type.

Value	Description
XA_AUDIOCODEC_PCM	PCM audio data.
XA_AUDIOCODEC_MP3	MPEG Layer III encoder.
XA_AUDIOCODEC_AMR	Adaptive Multi-Rate (AMR) speech encoder.
XA_AUDIOCODEC_AMRWB	Adaptive Multi-Rate Wideband (AMR-WB) speech encoder.
XA_AUDIOCODEC_AMRWBPLUS	Adaptive Multi-Rate Wideband Extended (AMR-WB+) speech encoder.
XA_AUDIOCODEC_AAC	MPEG4 Advanced Audio Coding.
XA_AUDIOCODEC_WMA	Windows Media Audio.
XA_AUDIOCODEC_REAL	Real Audio.
XA_AUDIOCODEC_VORBIS	Vorbis Audio.

9.2.4 XA_AUDIOPROFILE and XA_AUDIOMODE

```
#define XA_ AUDIOSTREAMFORMAT_UNDEFINED          ((XAuint32) 0x00000000)
```

Value	Description
XA_AUDIOSTREAMFORMAT_UNDEFINED	The codec does not have a stream format.

PCM Profiles and Modes

```
#define XA_AUDIOPROFILE_PCM                     ((XAuint32) 0x00000001)
```

The macros are used for defining the PCM audio profiles.

Value	Description
XA_AUDIOPROFILE_PCM	Default Profile for PCM encoded Audio

MP3 Profiles and Modes

```
#define XA_AUDIOPROFILE_MPEG1_L3                ((XAuint32) 0x00000001)
#define XA_AUDIOPROFILE_MPEG2_L3                ((XAuint32) 0x00000002)
#define XA_AUDIOPROFILE_MPEG25_L3               ((XAuint32) 0x00000003)

#define XA_AUDIOCHANMODE_MP3_MONO               ((XAuint32) 0x00000001)
#define XA_AUDIOCHANMODE_MP3_STEREO             ((XAuint32) 0x00000002)
#define XA_AUDIOCHANMODE_MP3_JOINTSTEREO        ((XAuint32) 0x00000003)
#define XA_AUDIOCHANMODE_MP3_DUAL               ((XAuint32) 0x00000004)
```

The macros are used for defining the MP3 audio profiles and modes.

Value	Description
XA_AUDIOPROFILE_MPEG1_L3	MPEG-1 Layer III.
XA_AUDIOPROFILE_MPEG2_L3	MPEG-2 Layer III.
XA_AUDIOPROFILE_MPEG25_L3	MPEG-2.5 Layer III.
XA_AUDIOCHANMODE_MP3_MONO	MP3 Mono mode.
XA_AUDIOCHANMODE_MP3_STEREO	MP3 Stereo Mode.
XA_AUDIOCHANMODE_MP3_JOINTSTEREO	MP3 Joint Stereo mode.
XA_AUDIOCHANMODE_MP3_DUAL	MP3 Dual Stereo mode.

AMR Profiles and Modes

```
#define XA_AUDIOPROFILE_AMR                    ((XAuint32) 0x00000001)

#define XA_AUDIOSTREAMFORMAT_CONFORMANCE       ((XAuint32) 0x00000001)
#define XA_AUDIOSTREAMFORMAT_IF1               ((XAuint32) 0x00000002)
#define XA_AUDIOSTREAMFORMAT_IF2               ((XAuint32) 0x00000003)
#define XA_AUDIOSTREAMFORMAT_FSF               ((XAuint32) 0x00000004)
#define XA_AUDIOSTREAMFORMAT_RTPPAYLOAD        ((XAuint32) 0x00000005)
#define XA_AUDIOSTREAMFORMAT_ITU               ((XAuint32) 0x00000006)
```

The macros are used for defining the AMR audio profiles and modes.

Value	Description
XA_AUDIOPROFILE_AMR	Adaptive Multi-Rate audio codec.
XA_AUDIOSTREAMFORMAT_CONFORMANCE	Standard test-sequence format.
XA_AUDIOSTREAMFORMAT_IF1	Interface format 1.
XA_AUDIOSTREAMFORMAT_IF2	Interface format 2.
XA_AUDIOSTREAMFORMAT_FSF	File Storage format.
XA_AUDIOSTREAMFORMAT_RTPPAYLOAD	RTP payload format.
XA_AUDIOSTREAMFORMAT_ITU	ITU frame format.

AMR-WB Profiles and Modes

```
#define XA_AUDIOPROFILE_AMRWB    ((XAuint32) 0x00000001)
```

The macros are used for defining the AMR-WB audio profiles.

Value	Description
XA_AUDIOPROFILE_AMRWB	Adaptive Multi-Rate - Wideband.

AMR-WB+ Profiles and Modes

```
#define XA_AUDIOPROFILE_AMRWBPLUS      ((XAuint32) 0x00000001)
```

The macros are used for defining the AMR-WB+ audio profiles.

Value	Description
XA_AUDIOPROFILE_AMRWBPLUS	Extended Adaptive Multi-Rate – Wideband.

AAC Profiles and Modes

```
#define XA_AUDIOPROFILE_AAC_AAC         ((XAuint32) 0x00000001)

#define XA_AUDIOMODE_AAC_MAIN           ((XAuint32) 0x00000001)
#define XA_AUDIOMODE_AAC_LC             ((XAuint32) 0x00000002)
#define XA_AUDIOMODE_AAC_SSR            ((XAuint32) 0x00000003)
#define XA_AUDIOMODE_AAC_LTP            ((XAuint32) 0x00000004)
#define XA_AUDIOMODE_AAC_HE             ((XAuint32) 0x00000005)
#define XA_AUDIOMODE_AAC_SCALABLE       ((XAuint32) 0x00000006)
#define XA_AUDIOMODE_AAC_ERLC           ((XAuint32) 0x00000007)
#define XA_AUDIOMODE_AAC_LD             ((XAuint32) 0x00000008)
#define XA_AUDIOMODE_AAC_HE_PS          ((XAuint32) 0x00000009)
#define XA_AUDIOMODE_AAC_HE_MPS         ((XAuint32) 0x0000000A)

#define XA_AUDIOSTREAMFORMAT_MP2ADTS    ((XAuint32) 0x00000001)
#define XA_AUDIOSTREAMFORMAT_MP4ADTS    ((XAuint32) 0x00000002)
#define XA_AUDIOSTREAMFORMAT_MP4LOAS    ((XAuint32) 0x00000003)
#define XA_AUDIOSTREAMFORMAT_MP4LATM    ((XAuint32) 0x00000004)
#define XA_AUDIOSTREAMFORMAT_ADIF       ((XAuint32) 0x00000005)
#define XA_AUDIOSTREAMFORMAT_MP4FF      ((XAuint32) 0x00000006)
#define XA_AUDIOSTREAMFORMAT_RAW        ((XAuint32) 0x00000007)
```

The macros are used for defining the AAC audio profiles and modes.

Value	Description
XA_AUDIOPROFILE_AAC_AAC	Advanced Audio Coding.
XA_AUDIOMODE_AAC_MAIN	AAC Main Profile.
XA_AUDIOMODE_AAC_LC	AAC Low Complexity.
XA_AUDIOMODE_AAC_SSR	AAC Scalable Sample Rate.
XA_AUDIOMODE_AAC_LTP	ACC Long Term Prediction.
XA_AUDIOMODE_AAC_HE	AAC High Efficiency.
XA_AUDIOMODE_AAC_SCALABLE	AAC Scalable.
XA_AUDIOMODE_AAC_ERLC	AAC Error Resilient LC.
XA_AUDIOMODE_AAC_LD	AAC Low Delay.
XA_AUDIOMODE_AAC_HE_PS	AAC High Efficiency with Parametric Stereo Coding.
XA_AUDIOMODE_AAC_HE_MPS	AAC High Efficiency with MPEG Surround Coding.
XA_AUDIOSTREAMFORMAT_MP2ADTS	MPEG-2 AAC Audio Data Transport Stream format.
XA_AUDIOSTREAMFORMAT_MP4ADTS	MPEG-4 AAC Audio Data Transport Stream format.
XA_AUDIOSTREAMFORMAT_MP4LOAS	Low Overhead Audio Stream format.
XA_AUDIOSTREAMFORMAT_MP4LATM	Low Overhead Audio Transport Multiplex.
XA_AUDIOSTREAMFORMAT_ADIF	Audio Data Interchange Format.
XA_AUDIOSTREAMFORMAT_MP4FF	AAC inside MPEG-4/ISO File Format.
XA_AUDIOSTREAMFORMAT_RAW	AAC Raw Format (access units).

Windows Media Audio Profiles and Modes

```
#define XA_AUDIOPROFILE_WMA7            ((XAuint32) 0x00000001)
#define XA_AUDIOPROFILE_WMA8            ((XAuint32) 0x00000002)
#define XA_AUDIOPROFILE_WMA9            ((XAuint32) 0x00000003)
#define XA_AUDIOPROFILE_WMA10           ((XAuint32) 0x00000004)

#define XA_AUDIOMODE_WMA_LEVEL1         ((XAuint32) 0x00000001)
#define XA_AUDIOMODE_WMA_LEVEL2         ((XAuint32) 0x00000002)
#define XA_AUDIOMODE_WMA_LEVEL3         ((XAuint32) 0x00000003)
#define XA_AUDIOMODE_WMA_LEVEL4         ((XAuint32) 0x00000004)
#define XA_AUDIOMODE_WMAPRO_LEVELM0     ((XAuint32) 0x00000005)
#define XA_AUDIOMODE_WMAPRO_LEVELM1     ((XAuint32) 0x00000006)
#define XA_AUDIOMODE_WMAPRO_LEVELM2     ((XAuint32) 0x00000007)
#define XA_AUDIOMODE_WMAPRO_LEVELM3     ((XAuint32) 0x00000008)
```

The macros are used for defining the WMA audio profiles and modes.

Value	Description
XA_AUDIOPROFILE_WMA7	Windows Media Audio Encoder V7.
XA_AUDIOPROFILE_WMA8	Windows Media Audio Encoder V8.
XA_AUDIOPROFILE_WMA9	Windows Media Audio Encoder V9.
XA_AUDIOPROFILE_WMA10	Windows Media Audio Encoder V10.
XA_AUDIOMODE_WMA_LEVEL1	WMA Level 1.
XA_AUDIOMODE_WMA_LEVEL2	WMA Level 2.
XA_AUDIOMODE_WMA_LEVEL3	WMA Level 3.
XA_AUDIOMODE_WMA_LEVEL3	WMA Level 4.
XA_AUDIOMODE_WMAPRO_LEVELM0	WMA Pro Level M0.
XA_AUDIOMODE_WMAPRO_LEVELM1	WMA Pro Level M1.
XA_AUDIOMODE_WMAPRO_LEVELM2	WMA Pro Level M2.
XA_AUDIOMODE_WMAPRO_LEVELM3	WMA Pro Level M3.

RealAudio Profiles and Levels

```
#define XA_AUDIOPROFILE_REALAUDIO            ((XAuint32) 0x00000001)

#define XA_AUDIOMODE_REALAUDIO_G2            ((XAuint32) 0x00000001)
#define XA_AUDIOMODE_REALAUDIO_8             ((XAuint32) 0x00000002)
#define XA_AUDIOMODE_REALAUDIO_10            ((XAuint32) 0x00000003)
#define XA_AUDIOMODE_REALAUDIO_SURROUND      ((XAuint32) 0x00000004)
```

The macros are used for defining the Real Audio audio profiles and modes.

Value	Description
XA_AUDIOPROFILE_REALAUDIO	RealAudio Encoder.
XA_AUDIOMODE_REALAUDIO_G2	RealAudio G2.
XA_AUDIOMODE_REALAUDIO_8	RealAudio 8.
XA_AUDIOMODE_REALAUDIO_10	RealAudio 10.
XA_AUDIOMODE_REALAUDIO_SURROUND	RealAudio Surround.

Vorbis Profiles and Levels

```
#define XA_AUDIOPROFILE_VORBIS                ((XAuint32) 0x00000001)

#define XA_AUDIOMODE_VORBIS                   ((XAuint32) 0x00000001)
```

The macros are used for defining the Vorbis audio profiles and modes.

Value	Description
XA_AUDIOPROFILE_VORBIS	Vorbis Encoder.
XA_AUDIOMODE_VORBIS	Default mode for Vorbis encoded audio.

9.2.5 XA_BOOLEAN

```
#define XA_BOOLEAN_FALSE   ((XAboolean) 0x00000000)
#define XA_BOOLEAN_TRUE    ((XAboolean) 0x00000001)
```

Canonical values for Boolean type.

Value	Description
XA_BOOLEAN_FALSE	False value for `XAboolean`.
XA_BOOLEAN_TRUE	True value for `XAboolean`.

9.2.6 XA_BYTEORDER

```
#define XA_BYTEORDER_BIGENDIAN          ((XAuint32) 0x00000001)
#define XA_BYTEORDER_LITTLEENDIAN       ((XAuint32) 0x00000002)
#define XA_BYTEORDER_NATIVE             <system dependent>
```

XA_BYTEORDER represents the byte order of a block of 16-bit, 32-bit or 64-bit data.

Value	Description
XA_BYTEORDER_BIGENDIAN	Big-endian data
XA_BYTEORDER_LITTLEENDIAN	Little-endian data
XA_BYTEORDER_NATIVE	Either big or little endian, based on system configuration via XA_BYTEORDER_NATIVEBIGENDIAN

9.2.7 XA_CAMERA_APERTUREMODE

```
#define XA_CAMERA_APERTUREMODE_MANUAL  ((XAuint32) 0x00000001)
#define XA_CAMERA_APERTUREMODE_AUTO    ((XAuint32) 0x00000002)
```

These values are used to set camera aperture setting.

Value	Description
XA_CAMERA_APERTUREMODE_MANUAL	Manual aperture mode
XA_CAMERA_APERTUREMODE_AUTO	Auto aperture mode

9.2.8 XA_CAMERA_AUTOEXPOSURESTATUS

```
#define XA_CAMERA_AUTOEXPOSURESTATUS_SUCCESS       ((XAuint32) 0x00000001)
#define XA_CAMERA_AUTOEXPOSURESTATUS_UNDEREXPOSURE ((XAuint32) 0x00000002)
#define XA_CAMERA_AUTOEXPOSURESTATUS_OVEREXPOSURE  ((XAuint32) 0x00000003)
```

These values represent different statuses of the automatic exposure.

Value	Description
XA_CAMERA_AUTOEXPOSURESTATUS_SUCCESS	Auto exposure has been successfully locked.
XA_CAMERA_AUTOEXPOSURESTATUS_UNDEREXPOSURE	Auto exposure has been locked, but the photo will we underexposed in the current lighting conditions. Consider changing manually the exposure settings or freeing the lock and trying the locking again.
XA_CAMERA_AUTOEXPOSURESTATUS_OVEREXPOSURE	Auto exposure has been locked, but the photo will we overexposed in the current lighting conditions. Consider changing manually the exposure settings or freeing the lock and trying the locking again.

9.2.9 XA_CAMERACBEVENT

```
#define XA_CAMERACBEVENT_ROTATION             ((XAuint32) 0x00000001)
#define XA_CAMERACBEVENT_FLASHREADY           ((XAuint32) 0x00000002)
#define XA_CAMERACBEVENT_FOCUSSTATUS          ((XAuint32) 0x00000003)
#define XA_CAMERACBEVENT_EXPOSURESTATUS       ((XAuint32) 0x00000004)
#define XA_CAMERACBEVENT_WHITEBALANCELOCKED   ((XAuint32) 0x00000005)
#define XA_CAMERACBEVENT_ZOOMSTATUS           ((XAuint32) 0x00000006)
```

These values are used to identify the callback event type.

Value	Description
XA_CAMERACBEVENT_ROTATION	This event indicates that a change in the camera's rotation setting has occurred. The `eventData` parameter will specify the new rotation setting. A rotation value of 0 degrees indicates the camera is un-rotated. A value of 0xFFFFFFFF will be returned if the camera rotation can not be determined
XA_CAMERACBEVENT_FLASHREADY	This event indicates that the flash is ready for use. The `eventData` parameter for this event is not used and shall be ignored.
XA_CAMERACBEVENT_FOCUSSTATUS	This event indicates that focusing is completed. The `eventData` parameter contains the focus status, see `XA_CAMERA_FOCUSMODESTATUS`.
XA_CAMERACBEVENT_EXPOSURESTATUS	This event indicates that locking of the auto exposure is completed. The eventData parameter contains the exposure status, see XA_CAMERA_AUTOEXPOSURESTATUS.
XA_CAMERACBEVENT_WHITEBALANCELOCKED	This event indicates that locking of the automatic white balance is completed. The eventData parameter for this event is not used and shall be ignored.
XA_CAMERACBEVENT_ZOOMSTATUS	This event indicates that zooming is completed. The eventData parameter contains the zoom setting in permille units.

9.2.10 XA_CAMERACAP

```
#define XA_CAMERACAP_FLASH                  ((XAuint32) 0x00000001)
#define XA_CAMERACAP_AUTOFOCUS              ((XAuint32) 0x00000002)
#define XA_CAMERACAP_CONTINUOUSAUTOFOCUS    ((XAuint32) 0x00000004)
#define XA_CAMERACAP_MANUALFOCUS            ((XAuint32) 0x00000008)
#define XA_CAMERACAP_AUTOEXPOSURE           ((XAuint32) 0x00000010)
#define XA_CAMERACAP_MANUALEXPOSURE         ((XAuint32) 0x00000020)
#define XA_CAMERACAP_AUTOISOSENSITIVITY     ((XAuint32) 0x00000040)
#define XA_CAMERACAP_MANUALISOSENSITIVITY   ((XAuint32) 0x00000080)
#define XA_CAMERACAP_AUTOAPERTURE           ((XAuint32) 0x00000100)
#define XA_CAMERACAP_MANUALAPERTURE         ((XAuint32) 0x00000200)
#define XA_CAMERACAP_AUTOSHUTTERSPEED       ((XAuint32) 0x00000400)
#define XA_CAMERACAP_MANUALSHUTTERSPEED     ((XAuint32) 0x00000800)
#define XA_CAMERACAP_AUTOWHITEBALANCE       ((XAuint32) 0x00001000)
#define XA_CAMERACAP_MANUALWHITEBALANCE     ((XAuint32) 0x00002000)
#define XA_CAMERACAP_OPTICALZOOM            ((XAuint32) 0x00004000)
#define XA_CAMERACAP_DIGITALZOOM            ((XAuint32) 0x00008000)
#define XA_CAMERACAP_METERING               ((XAuint32) 0x00010000)
#define XA_CAMERACAP_BRIGHTNESS             ((XAuint32) 0x00020000)
#define XA_CAMERACAP_CONTRAST               ((XAuint32) 0x00040000)
#define XA_CAMERACAP_GAMMA                  ((XAuint32) 0x00080000)
```

The XA_CAMERACAP macros are used for camera capabilities. See XACameraItf (see section 8.6) and XAImageControlItf (see section 8.15) for more information on individual features.

Value	Description
XA_CAMERACAP_FLASH	Flash supported.
XA_CAMERACAP_AUTOFOCUS	One-shot auto focus modes supported.
XA_CAMERACAP_CONTINUOUSAUTOFOCUS	Continuous auto focus mode supported.
XA_CAMERACAP_MANUALFOCUS	Manual focus supported.
XA_CAMERACAP_AUTOEXPOSURE	Auto exposure algorithms supported.
XA_CAMERACAP_MANUALEXPOSURE	Manual exposure setting supported.
XA_CAMERACAP_AUTOSENSITIVITY	Auto sensitivity mode supported.
XA_CAMERACAP_MANUALSENSITIVITY	Manual sensitivity setting supported.
XA_CAMERACAP_AUTOAPERTURE	Auto aperture mode supported.
XA_CAMERACAP_MANUALAPERTURE	Manual aperture setting supported.
XA_CAMERACAP_AUTOSHUTTERSPEED	Auto shutter speed mode supported.
XA_CAMERACAP_MANUALSHUTTERSPEED	Manual shutter speed supported.
XA_CAMERACAP_AUTOWHITEBALANCE	Auto white balance modes supported.
XA_CAMERACAP_MANUALWHITEBALANCE	Manual white balance des supported.
XA_CAMERACAP_OPTICALZOOM	Camera supports optical zoom.
XA_CAMERACAP_DIGITALZOOM	Camera supports digital zoom.
XA_CAMREACAP_METERING	Exposure metering supported.
XA_CAMERACAP_BRIGHTNESS	Camera supports brightness controls.

Value	Description
XA_CAMERACAP_CONTRAST	Camera supports contrast controls.
XA_CAMERACAP_GAMMA	Camera supports gamma controls.

9.2.11 XA_CAMERA_EXPOSUREMODE

```
#define XA_CAMERA_EXPOSUREMODE_MANUAL         ((XAuint32) 0x00000001)
#define XA_CAMERA_EXPOSUREMODE_AUTO           ((XAuint32) 0x00000002)
#define XA_CAMERA_EXPOSUREMODE_NIGHT          ((XAuint32) 0x00000004)
#define XA_CAMERA_EXPOSUREMODE_BACKLIGHT      ((XAuint32) 0x00000008)
#define XA_CAMERA_EXPOSUREMODE_SPOTLIGHT      ((XAuint32) 0x00000010)
#define XA_CAMERA_EXPOSUREMODE_SPORTS         ((XAuint32) 0x00000020)
#define XA_CAMERA_EXPOSUREMODE_SNOW           ((XAuint32) 0x00000040)
#define XA_CAMERA_EXPOSUREMODE_BEACH          ((XAuint32) 0x00000080)
#define XA_CAMERA_EXPOSUREMODE_LARGEAPERTURE  ((XAuint32) 0x00000100)
#define XA_CAMERA_EXPOSUREMODE_SMALLAPERTURE  ((XAuint32) 0x00000200)
#define XA_CAMERA_EXPOSUREMODE_PORTRAIT       ((XAuint32) 0x00000400)
#define XA_CAMERA_EXPOSUREMODE_NIGHTPORTRAIT  ((XAuint32) 0x00000800)
```

These values are used to set camera exposure.

Value	Description
XA_CAMERA_EXPOSUREMODE_MANUAL	Manual exposure.
XA_CAMERA_EXPOSUREMODE_AUTO	Auto exposure mode.
XA_CAMERA_EXPOSUREMODE_NIGHT	Night exposure mode.
XA_CAMERA_EXPOSUREMODE_BACKLIGHT	Backlight exposure mode.
XA_CAMERA_EXPOSUREMODE_SPOTLIGHT	Spotlight exposure mode.
XA_CAMERA_EXPOSUREMODE_SPORTS	Spots exposure mode.
XA_CAMERA_EXPOSUREMODE_SNOW	Snow exposure mode.
XA_CAMERA_EXPOSUREMODE_BEACH	Beach exposure mode.
XA_CAMERA_EXPOSUREMODE_LARGEAPERTURE	Large aperture exposure mode.
XA_CAMERA_EXPOSUREMODE_SMALLAPERTURE	Small aperture exposure mode.
XA_CAMERA_EXPOSUREMODE_PORTRAIT	Portrait exposure mode.
XA_CAMERA_EXPOSUREMODE_NIGHTPORTRAIT	Night time portrait exposure mode.

9.2.12 XA_CAMERA_FLASHMODE

```
#define XA_CAMERA_FLASHMODE_OFF                    ((XAuint32) 0x00000001)
#define XA_CAMERA_FLASHMODE_ON                     ((XAuint32) 0x00000002)
#define XA_CAMERA_FLASHMODE_AUTO                   ((XAuint32) 0x00000004)
#define XA_CAMERA_FLASHMODE_REDEYEREDUCTION        ((XAuint32) 0x00000008)
#define XA_CAMERA_FLASHMODE_REDEYEREDUCTION_AUTO   ((XAuint32) 0x00000010)
#define XA_CAMERA_FLASHMODE_FILLIN                 ((XAuint32) 0x00000020)
#define XA_CAMERA_FLASHMODE_TORCH                  ((XAuint32) 0x00000040)
```

These values are used to set camera flash mode.

Value	Description
XA_CAMERA_FLASHMODE_OFF	Flash disabled.
XA_CAMERA_FLASHMODE_ON	Flash enabled.
XA_CAMERA_FLASHMODE_AUTO	Auto flash mode.
XA_CAMERA_FLASHMODE_REDEYEREDUCTION	Red eye reduction flash.
XA_CAMERA_FLASHMODE_REDEYEREDUCTION_AUTO	Red eye reduction flash automatic.
XA_CAMERA_FLASHMODE_FILLIN	Use flash to fill-in dimly lit areas.
XA_CAMERA_FLASHMODE_TORCH	Flash is always on.

9.2.13 XA_CAMERA_FOCUSMODE

```
#define XA_CAMERA_FOCUSMODE_MANUAL                  ((XAuint32) 0x00000001)
#define XA_CAMERA_FOCUSMODE_AUTO                    ((XAuint32) 0x00000002)
#define XA_CAMERA_FOCUSMODE_CENTROID                ((XAuint32) 0x00000004)
#define XA_CAMERA_FOCUSMODE_CONTINUOUS_AUTO         ((XAuint32) 0x00000008)
#define XA_CAMERA_FOCUSMODE_CONTINUOUS_CENTROID     ((XAuint32) 0x00000010)
```

These values are used to set camera focus mode.

Value	Description
XA_CAMERA_FOCUSMODE_MANUAL	Manual focus mode.
XA_CAMERA_FOCUSMODE_AUTO	One-shot auto focus mode. This mode, sometimes called also as “single auto focus”, automatically adjusts the focus once when it has been activated. Use XACameraItf::SetAutoLocks with parameter XA_CAMERA_LOCK_AUTOFOCUS to activate and lock the auto focusing. The lock is freed by clearing the XA_CAMERA_LOCK_AUTOFOCUS bit.
XA_CAMERA_FOCUSMODE_CENTROID	One-shot centroid auto focus mode.
XA_CAMERA_FOCUSMODE_CONTINUOUS_AUTO	Continuous auto focus mode. This mode, sometimes called also as “AF Servo”, continually adjusts the focus as long as the mode is active. When this mode is selected, use XACameraItf::SetAutoLocks with parameter XA_CAMERA_LOCK_AUTOFOCUS to activate the continuous focusing. The continuous focusing deactivates once XA_CAMERA_LOCK_AUTOFOCUS bit is cleared.
XA_CAMERA_FOCUSMODE_CONTINUOUS_CENTROID	Continuous centroid focus mode.

It is to be noted that not all cameras will be able to provide focus status events in continuous focusing mode. But this feature is quite useful for those cameras that do have this capability.

9.2.14 XA_CAMERA_FOCUSMODESTATUS

```
#define XA_CAMERA_FOCUSMODESTATUS_OFF               ((XAuint32) 0x00000001)
#define XA_CAMERA_FOCUSMODESTATUS_REQUEST           ((XAuint32) 0x00000002)
#define XA_CAMERA_FOCUSMODESTATUS_REACHED           ((XAuint32) 0x00000003)
#define XA_CAMERA_FOCUSMODESTATUS_UNABLETOREACH     ((XAuint32) 0x00000004)
#define XA_CAMERA_FOCUSMODESTATUS_LOST              ((XAuint32) 0x00000005)
```

These values are used to set camera focus mode.

Value	Description
XA_CAMERA_FOCUSMODESTATUS_OFF	Manual focus mode is in use, focus status is not available.
XA_CAMERA_FOCUSMODESTATUS_REQUEST	Focus request is in progress.
XA_CAMERA_FOCUSMODESTATUS_REACHED	Focus has been reached.
XA_CAMERA_FOCUSMODESTATUS_UNABLETOREACH	Unable to achieve focus.
XA_CAMERA_FOCUSMODESTATUS_LOST	Focus has been lost.

9.2.15 XA_CAMERA_ISOSENSITIVITYMODE

```
#define XA_CAMERA_ISOSENSITIVITYMODE_MANUAL  ((XAuint32) 0x00000001)
#define XA_CAMERA_ISOSENSITIVITYMODE_AUTO    ((XAuint32) 0x00000002)
```

These values are used to set camera ISO sensitivity.

Value	Description
XA_CAMERA_ISOSENSITIVITYMODE_MANUAL	Manual sensitivity mode.
XA_CAMERA_ISOSENSITIVITYMODE_AUTO	Auto sensitivity mode.

9.2.16 XA_CAMERA_LOCK

```
#define XA_CAMERA_LOCK_AUTOFOCUS             ((XAuint32) 0x00000001)
#define XA_CAMERA_LOCK_AUTOEXPOSURE          ((XAuint32) 0x00000002)
#define XA_CAMERA_LOCK_AUTOWHITEBALANCE      ((XAuint32) 0x00000004)
```

These values are used to refer to various locks of the automatic camera settings.

Value	Description
XA_CAMERA_LOCK_AUTOFOCUS	Lock for the automatic focus.
XA_CAMERA_LOCK_AUTOEXPOSURE	Lock for the automatic exposure settings.
XA_CAMERA_LOCK_AUTOWHITEBALANCE	Lock for the automatic white balance.

9.2.17 XA_CAMERA_METERINGMODE

```
#define XA_CAMERA_METERINGMODE_AVERAGE ((XAuint32) 0x00000001)
#define XA_CAMERA_METERINGMODE_SPOT    ((XAuint32) 0x00000002)
#define XA_CAMERA_METERINGMODE_MATRIX  ((XAuint32) 0x00000004)
```

These values are used to set camera metering mode for exposure.

Value	Description
XA_CAMERA_METERINGMODE_AVERAGE	Center weighted average metering mode.
XA_CAMERA_METERINGMODE_SPOT	Spot (partial) metering mode.
XA_CAMERA_METERINGMODE_MATRIX	Matrix or evaluative metering mode.

9.2.18 XA_CAMERA_SHUTTERSPEEDMODE

```
#define XA_CAMERA_SHUTTERSPEEDMODE_MANUAL    ((XAuint32) 0x00000001)
#define XA_CAMERA_SHUTTERSPEEDMODE_AUTO      ((XAuint32) 0x00000002)
```

These values are used to set camera shutter speed.

Value	Description
XA_CAMERA_SHUTTERSPEEDMODE_MANUAL	Manual shutter speed mode.
XA_CAMERA_SHUTTERSPEEDMODE_AUTO	Auto shutter speed mode.

9.2.19 XA_CAMERA_WHITEBALANCEMODE

```
#define XA_CAMERA_WHITEBALANCEMODE_MANUAL          ((XAuint32) 0x00000001)
#define XA_CAMERA_WHITEBALANCEMODE_AUTO            ((XAuint32) 0x00000002)
#define XA_CAMERA_WHITEBALANCEMODE_SUNLIGHT        ((XAuint32) 0x00000004)
#define XA_CAMERA_WHITEBALANCEMODE_CLOUDY          ((XAuint32) 0x00000008)
#define XA_CAMERA_WHITEBALANCEMODE_SHADE           ((XAuint32) 0x00000010)
#define XA_CAMERA_WHITEBALANCEMODE_TUNGSTEN        ((XAuint32) 0x00000020)
#define XA_CAMERA_WHITEBALANCEMODE_FLUORESCENT     ((XAuint32) 0x00000040)
#define XA_CAMERA_WHITEBALANCEMODE_INCANDESCENT    ((XAuint32) 0x00000080)
#define XA_CAMERA_WHITEBALANCEMODE_FLASH           ((XAuint32) 0x00000100)
#define XA_CAMERA_WHITEBALANCEMODE_SUNSET          ((XAuint32) 0x00000200)
```

These values are used to set camera white balance.

Value	Description
XA_CAMERA_WHITEBALANCEMODE_MANUAL	White balance off.
XA_CAMERA_WHITEBALANCEMODE_AUTO	Auto white balance mode.
XA_CAMERA_WHITEBALANCEMODE_SUNLIGHT	Sunlight white balance mode.
XA_CAMERA_WHITEBALANCEMODE_CLOUDY	Cloudy white balance mode.
XA_CAMERA_WHITEBALANCEMODE_SHADE	Shade white balance mode.
XA_CAMERA_WHITEBALANCEMODE_TUNGSTEN	Tungsten white balance mode.
XA_CAMERA_WHITEBALANCEMODE_FLUORESCENT	Fluorescent white balance mode.
XA_CAMERA_WHITEBALANCEMODE_INCANDESCENT	Incandescent white balance mode.
XA_CAMERA_WHITEBALANCEMODE_FLASH	Flash white balance mode.
XA_CAMERA_WHITEBALANCEMODE_SUNSET	Sunset white balance mode.

9.2.20 XA_CAMERA_ZOOM

```
#define XA_CAMERA_ZOOM_SLOW      ((XAuint32) 50)
#define XA_CAMERA_ZOOM_NORMAL    ((XAuint32) 100)
#define XA_CAMERA ZOOM_FAST      ((XAuint32) 200)
#define XA_CAMERA ZOOM_FASTEST   ((XAuint32) 0xFFFFFFFF)
```

These values are used to hint camera zooming speed with method XACameraItf::SetZoom.

Value	Description
XA_CAMERA_ZOOM_SLOW	Slow zooming speed.
XA_CAMERA_ZOOM_NORMAL	Normal zooming speed.
XA_CAMERA_ZOOM_FAST	Fast zooming speed.
XA_CAMERA_ZOOM_FASTEST	Fastest zooming speed to be used if the application prefers immediate action.

9.2.21 XA_CHARACTERENCODING

```
#define XA_CHARACTERENCODING_UNKNOWN                ((XAuint32) 0x00000000)
#define XA_CHARACTERENCODING_BINARY                 ((XAuint32) 0x00000001)
#define XA_CHARACTERENCODING_ASCII                  ((XAuint32) 0x00000002)
#define XA_CHARACTERENCODING_BIG5                   ((XAuint32) 0x00000003)
#define XA_CHARACTERENCODING_CODEPAGE1252           ((XAuint32) 0x00000004)
#define XA_CHARACTERENCODING_GB2312                 ((XAuint32) 0x00000005)
#define XA_CHARACTERENCODING_HZGB2312               ((XAuint32) 0x00000006)
#define XA_CHARACTERENCODING_GB12345                ((XAuint32) 0x00000007)
#define XA_CHARACTERENCODING_GB18030                ((XAuint32) 0x00000008)
#define XA_CHARACTERENCODING_GBK                    ((XAuint32) 0x00000009)
#define XA_CHARACTERENCODING_IMAPUTF7               ((XAuint32) 0x0000000A)
#define XA_CHARACTERENCODING_ISO2022JP              ((XAuint32) 0x0000000B)
#define XA_CHARACTERENCODING_ISO2022JP1             ((XAuint32) 0x0000000B)
#define XA_CHARACTERENCODING_ISO88591               ((XAuint32) 0x0000000C)
#define XA_CHARACTERENCODING_ISO885910              ((XAuint32) 0x0000000D)
#define XA_CHARACTERENCODING_ISO885913              ((XAuint32) 0x0000000E)
#define XA_CHARACTERENCODING_ISO885914              ((XAuint32) 0x0000000F)
#define XA_CHARACTERENCODING_ISO885915              ((XAuint32) 0x00000010)
#define XA_CHARACTERENCODING_ISO88592               ((XAuint32) 0x00000011)
#define XA_CHARACTERENCODING_ISO88593               ((XAuint32) 0x00000012)
#define XA_CHARACTERENCODING_ISO88594               ((XAuint32) 0x00000013)
#define XA_CHARACTERENCODING_ISO88595               ((XAuint32) 0x00000014)
#define XA_CHARACTERENCODING_ISO88596               ((XAuint32) 0x00000015)
#define XA_CHARACTERENCODING_ISO88597               ((XAuint32) 0x00000016)
#define XA_CHARACTERENCODING_ISO88598               ((XAuint32) 0x00000017)
#define XA_CHARACTERENCODING_ISO88599               ((XAuint32) 0x00000018)
#define XA_CHARACTERENCODING_ISOEUCJP               ((XAuint32) 0x00000019)
#define XA_CHARACTERENCODING_SHIFTJIS               ((XAuint32) 0x0000001A)
#define XA_CHARACTERENCODING_SMS7BIT                ((XAuint32) 0x0000001B)
#define XA_CHARACTERENCODING_UTF7                   ((XAuint32) 0x0000001C)
#define XA_CHARACTERENCODING_UTF8                   ((XAuint32) 0x0000001D)
#define XA_CHARACTERENCODING_JAVACONFORMANTUTF8     ((XAuint32) 0x0000001E)
#define XA_CHARACTERENCODING_UTF16BE                ((XAuint32) 0x0000001F)
#define XA_CHARACTERENCODING_UTF16LE                ((XAuint32) 0x00000020)
```

XA_CHARACTERENCODING represents a character encoding for metadata keys and values.

Value	Description
XA_CHARACTERENCODING_UNKNOWN	Unknown character encoding.
XA_CHARACTERENCODING_BINARY	Binary data.
XA_CHARACTERENCODING_ASCII	ASCII.
XA_CHARACTERENCODING_BIG5	Big 5.
XA_CHARACTERENCODING_CODEPAGE1252	Microsoft Code Page 1252.
XA_CHARACTERENCODING_GB2312	GB 2312 (Chinese).
XA_CHARACTERENCODING_HZGB2312	HZ GB 2312 (Chinese).
XA_CHARACTERENCODING_GB12345	GB 12345 (Chinese).
XA_CHARACTERENCODING_GB18030	GB 18030 (Chinese).
XA_CHARACTERENCODING_GBK	GBK (CP936) (Chinese).
XA_CHARACTERENCODING_ISO2022JP	ISO-2022-JP (Japanese).
XA_CHARACTERENCODING_ISO2022JP1	ISO-2022-JP-1 (Japanese).
XA_CHARACTERENCODING_ISO88591	ISO-8859-1 (Latin-1).
XA_CHARACTERENCODING_ISO88592	ISO-8859-1 (Latin-2).
XA_CHARACTERENCODING_ISO88593	ISO-8859-1 (Latin-3).
XA_CHARACTERENCODING_ISO88594	ISO-8859-1 (Latin-4).
XA_CHARACTERENCODING_ISO88595	ISO-8859-1 (Latin/Cyrillic).
XA_CHARACTERENCODING_ISO88596	ISO-8859-1 (Latin/Arabic).
XA_CHARACTERENCODING_ISO88597	ISO-8859-1 (Latin/Greek).
XA_CHARACTERENCODING_ISO88598	ISO-8859-1 (Latin/Hebrew).
XA_CHARACTERENCODING_ISO88599	ISO-8859-1 (Latin-5).
XA_CHARACTERENCODING_ISO885910	ISO-8859-1 (Latin-6).
XA_CHARACTERENCODING_ISO885913	ISO-8859-1 (Latin-7).
XA_CHARACTERENCODING_ISO885914	ISO-8859-1 (Latin-8).
XA_CHARACTERENCODING_ISO885915	ISO-8859-1 (Latin-9).
XA_CHARACTERENCODING_ISOEUCJP	ISO EUC-JP.
XA_CHARACTERENCODING_SHIFTJIS	Shift-JIS (Japanese).
XA_CHARACTERENCODING_SMS7BIT	SMS 7-bit.
XA_CHARACTERENCODING_UTF7	Unicode UTF-7.
XA_CHARACTERENCODING_IMAPUTF7	Unicode UTF-7 per IETF RFC 2060.
XA_CHARACTERENCODING_UTF8	Unicode UTF-8.
XA_CHARACTERENCODING_JAVACONFORMANTUTF8	Unicode UTF-8 (Java Conformant).
XA_CHARACTERENCODING_UTF16BE	Unicode UTF-16 (Big Endian).
XA_CHARACTERENCODING_UTF16LE	Unicode UTF-16 (Little Endian).

9.2.22 XA_COLORFORMAT

```
#define XA_COLORFORMAT_UNUSED                    ((XAuint32) 0x00000000)
#define XA_COLORFORMAT_MONOCHROME                ((XAuint32) 0x00000001)
#define XA_COLORFORMAT_8BITRGB332                ((XAuint32) 0x00000002)
#define XA_COLORFORMAT_12BITRGB444               ((XAuint32) 0x00000003)
#define XA_COLORFORMAT_16BITARGB4444             ((XAuint32) 0x00000004)
#define XA_COLORFORMAT_16BITARGB1555             ((XAuint32) 0x00000005)
#define XA_COLORFORMAT_16BITRGB565               ((XAuint32) 0x00000006)
#define XA_COLORFORMAT_16BITBGR565               ((XAuint32) 0x00000007)
#define XA_COLORFORMAT_18BITRGB666               ((XAuint32) 0x00000008)
#define XA_COLORFORMAT_18BITARGB1665             ((XAuint32) 0x00000009)
#define XA_COLORFORMAT_19BITARGB1666             ((XAuint32) 0x0000000A)
#define XA_COLORFORMAT_24BITRGB888               ((XAuint32) 0x0000000B)
#define XA_COLORFORMAT_24BITBGR888               ((XAuint32) 0x0000000C)
#define XA_COLORFORMAT_24BITARGB1887             ((XAuint32) 0x0000000D)
#define XA_COLORFORMAT_25BITARGB1888             ((XAuint32) 0x0000000E)
#define XA_COLORFORMAT_32BITBGRA8888             ((XAuint32) 0x0000000F)
#define XA_COLORFORMAT_32BITARGB8888             ((XAuint32) 0x00000010)
#define XA_COLORFORMAT_YUV411PLANAR              ((XAuint32) 0x00000011)
#define XA_COLORFORMAT_YUV420PLANAR              ((XAuint32) 0x00000013)
#define XA_COLORFORMAT_YUV420SEMIPLANAR          ((XAuint32) 0x00000015)
#define XA_COLORFORMAT_YUV422PLANAR              ((XAuint32) 0x00000016)
#define XA_COLORFORMAT_YUV422SEMIPLANAR          ((XAuint32) 0x00000018)
#define XA_COLORFORMAT_YCBYCR                    ((XAuint32) 0x00000019)
#define XA_COLORFORMAT_YCRYCB                    ((XAuint32) 0x0000001A)
#define XA_COLORFORMAT_CBYCRY                    ((XAuint32) 0x0000001B)
#define XA_COLORFORMAT_CRYCBY                    ((XAuint32) 0x0000001C)
#define XA_COLORFORMAT_YUV444INTERLEAVED         ((XAuint32) 0x0000001D)
#define XA_COLORFORMAT_RAWBAYER8BIT              ((XAuint32) 0x0000001E)
#define XA_COLORFORMAT_RAWBAYER10BIT             ((XAuint32) 0x0000001F)
#define XA_COLORFORMAT_RAWBAYER8BITCOMPRESSED    ((XAuint32) 0x00000020)
#define XA_COLORFORMAT_L2                        ((XAuint32) 0x00000021)
#define XA_COLORFORMAT_L4                        ((XAuint32) 0x00000022)
#define XA_COLORFORMAT_L8                        ((XAuint32) 0x00000023)
#define XA_COLORFORMAT_L16                       ((XAuint32) 0x00000024)
#define XA_COLORFORMAT_L24                       ((XAuint32) 0x00000025)
#define XA_COLORFORMAT_L32                       ((XAuint32) 0x00000026)
#define XA_COLORFORMAT_18BITBGR666               ((XAuint32) 0x00000029)
#define XA_COLORFORMAT_24BITARGB6666             ((XAuint32) 0x0000002A)
#define XA_COLORFORMAT_24BITABGR6666             ((XAuint32) 0x0000002B)
```

These values are used to set pixel color formats.

Value	Description
XA_COLORFORMAT_UNUSED	Value used when color format is not used or applicable.
XA_COLORFORMAT_MONOCHROME	1 bit per pixel monochrome.
XA_COLORFORMAT_8BITRGB332	8 bits per pixel RGB format with colors stored as Red 7:5, Green 4:2, and Blue 1:0.
XA_COLORFORMAT_12BITRGB444	12 bits per pixel RGB format with colors stored as Red 11:8, Green 7:4, and Blue 3:0.
XA_COLORFORMAT_16BITARGB4444	16 bits per pixel ARGB format with colors stored as Alpha 15:12, Red 11:8, Green 7:4, and Blue 3:0.

Value	Description
XA_COLORFORMAT_16BITARGB1555	16 bits per pixel ARGB format with colors stored as Alpha 15, Red 14:10, Green 9:5, and Blue 4:0.
XA_COLORFORMAT_16BITRGB565	16 bits per pixel RGB format with colors stored as Red 15:11, Green 10:5, and Blue 4:0.
XA_COLORFORMAT_16BITBGR565	16 bits per pixel BGR format with colors stored as Blue 15:11, Green 10:5, and Red 4:0.
XA_COLORFORMAT_18BITRGB666	18 bits per pixel RGB format with colors stored as Red 17:12, Green 11:6, and Blue 5:0.
XA_COLORFORMAT_18BITARGB1665	18 bits per pixel ARGB format with colors stored as Alpha 17, Red 16:11, Green 10:5, and Blue 4:0.
XA_COLORFORMAT_19BITARGB1666	19 bits per pixel ARGB format with colors stored as Alpha 18, Red 17:12, Green 11:6, and Blue 5:0.
XA_COLORFORMAT_24BITRGB888	24 bits per pixel RGB format with colors stored as Red 23:16, Green 15:8, and Blue 7:0.
XA_COLORFORMAT_24BITBGR888	24 bits per pixel BGR format with colors stored as Blue 23:16, Green 15:8, and Red 7:0.
XA_COLORFORMAT_24BITARGB1887	24 bits per pixel ARGB format with colors stored as Alpha 23, Red 22:15, Green 14:7, and Blue 6:0.
XA_COLORFORMAT_25BITARGB1888	25 bits per pixel ARGB format with colors stored as Alpha 24, Red 23:16, Green 15:8, and Blue 7:0.
XA_COLORFORMAT_32BITBGRA8888	32 bits per pixel ARGB format with colors stored as Alpha 31:24 Red 23:16, Green 15:8, and Blue 7:0.
XA_COLORFORMAT_32BITARGB8888	24 bits per pixel ABGR format with colors stored as Alpha 31:24, Blue 23:16, Green 15:8, and Red 7:0.
XA_COLORFORMAT_YUV411PLANAR	YUV planar format, organized with three separate planes for each color component, namely Y, U, and V. U and V pixels are sub-sampled by a factor of four both horizontally and vertically.
XA_COLORFORMAT_YUV420PLANAR	YUV planar format, organized with three separate planes for each color component, namely Y, U, and V. U and V pixels are sub-sampled by a factor of two both horizontally and vertically.
XA_COLORFORMAT_YUV420SEMIPLANAR	YUV planar format, organized with a first plane containing Y pixels, and a second plane containing interleaved U and V pixels. U and V pixels are sub-sampled by a factor of two both horizontally and vertically.
XA_COLORFORMAT_YUV422PLANAR	YUV planar format, organized with three separate planes for each color component, namely Y, U, and V.
XA_COLORFORMAT_YUV422SEMIPLANAR	YUV planar format, organized with a first plane containing Y pixels and a second plane containing interleaved U and V pixels.
XA_COLORFORMAT_YCBYCR	16 bits per pixel YUV interleaved format organized as

Value	Description
	YUYV (i.e., YCbYCr).
XA_COLORFORMAT_YCRYCB	16 bits per pixel YUV interleaved format organized as YVYU (i.e., YCrYCb).
XA_COLORFORMAT_CBYCRY	16 bits per pixel YUV interleaved format organized as UYVY (i.e., CbYCrY).
XA_COLORFORMAT_CRYCBY	16 bits per pixel YUV interleaved format organized as VYUY (i.e., CrYCbY).
XA_COLORFORMAT_YUV444INTERLEAVED	12 bits per pixel YUV format with colors stores as Y 11:8, U 7:4, and V 3:0.
XA_COLORFORMAT_RAWBAYER8BIT	SMIA 8-bit raw Bayer pattern camera format.
XA_COLORFORMAT_RAWBAYER10BIT	SMIA 10-bit raw Bayer pattern camera format.
XA_COLORFORMAT_RAWBAYER8BITCOMPRESSED	SMIA compressed 8-bit camera output format.
XA_COLORFORMAT_L2	2 bit per pixel luminance.
XA_COLORFORMAT_L4	4 bit per pixel luminance.
XA_COLORFORMAT_L8	8 bit per pixel luminance.
XA_COLORFORMAT_L16	16 bit per pixel luminance.
XA_COLORFORMAT_L24	24 bit per pixel luminance.
XA_COLORFORMAT_L32	32 bit per pixel luminance.
XA_COLORFORMAT_18BITBGR666	18 bits per pixel BGR format with colors stored as Blue 17:12, Green 11:6, and Red 5:0.
XA_COLORFORMAT_24BITARGB6666	24 bits per pixel ARGB format with colors stored as Alpha 23:18, Red 17:12, Green 11:6, and Blue 5:0
XA_COLORFORMAT_24BITABGR6666	24 bits per pixel ARGB format with colors stored as Alpha 23:18, Blue 17:12, Green 11:6, and Red 5:0

9.2.23 XA_CONTAINERTYPE

```
#define XA_CONTAINERTYPE_UNSPECIFIED   ((XAuint32) 0x00000001)
#define XA_CONTAINERTYPE_RAW           ((XAuint32) 0x00000002)
#define XA_CONTAINERTYPE_ASF           ((XAuint32) 0x00000003)
#define XA_CONTAINERTYPE_AVI           ((XAuint32) 0x00000004)
#define XA_CONTAINERTYPE_BMP           ((XAuint32) 0x00000005)
#define XA_CONTAINERTYPE_JPG           ((XAuint32) 0x00000006)
#define XA_CONTAINERTYPE_JPG2000       ((XAuint32) 0x00000007)
#define XA_CONTAINERTYPE_M4A           ((XAuint32) 0x00000008)
#define XA_CONTAINERTYPE_MP3           ((XAuint32) 0x00000009)
#define XA_CONTAINERTYPE_MP4           ((XAuint32) 0x0000000A)
#define XA_CONTAINERTYPE_MPEG_ES       ((XAuint32) 0x0000000B)
#define XA_CONTAINERTYPE_MPEG_PS       ((XAuint32) 0x0000000C)
#define XA_CONTAINERTYPE_MPEG_TS       ((XAuint32) 0x0000000D)
#define XA_CONTAINERTYPE_QT            ((XAuint32) 0x0000000E)
#define XA_CONTAINERTYPE_WAV           ((XAuint32) 0x0000000F)
#define XA_CONTAINERTYPE_XMF_0         ((XAuint32) 0x00000010)
#define XA_CONTAINERTYPE_XMF_1         ((XAuint32) 0x00000011)
#define XA_CONTAINERTYPE_XMF_2         ((XAuint32) 0x00000012)
#define XA_CONTAINERTYPE_XMF_3         ((XAuint32) 0x00000013)
#define XA_CONTAINERTYPE_XMF_GENERIC   ((XAuint32) 0x00000014)
#define XA_CONTAINERTYPE_AMR           ((XAuint32) 0x00000015)
#define XA_CONTAINERTYPE_AAC           ((XAuint32) 0x00000016)
#define XA_CONTAINERTYPE_3GPP          ((XAuint32) 0x00000017)
#define XA_CONTAINERTYPE_3GA           ((XAuint32) 0x00000018)
#define XA_CONTAINERTYPE_RM            ((XAuint32) 0x00000019)
#define XA_CONTAINERTYPE_DMF           ((XAuint32) 0x0000001A)
#define XA_CONTAINERTYPE_SMF           ((XAuint32) 0x0000001B)
#define XA_CONTAINERTYPE_MOBILE_DLS    ((XAuint32) 0x0000001C)
#define XA_CONTAINERTYPE_0GG           ((XAuint32) 0x0000001D)
```

XA_CONTAINERTYPE represents the container type of the data source or sink.

Value	Description
XA_CONTAINERTYPE_UNSPECIFIED	The container type is not specified.
XA_CONTAINERTYPE_RAW	There is no container. Content is in raw form.
XA_CONTAINERTYPE_ASF	The container type is ASF.
XA_CONTAINERTYPE_AVI	The container type is AVI.
XA_CONTAINERTYPE_BMP	The container type is BMP.
XA_CONTAINERTYPE_JPG	The container type is JPEG.
XA_CONTAINERTYPE_JPG2000	The container type is JPEG 2000.
XA_CONTAINERTYPE_M4A	The container type is M4A.
XA_CONTAINERTYPE_MP3	The container type is MP3.
XA_CONTAINERTYPE_MP4	The container type is MP4.
XA_CONTAINERTYPE_MPEG_ES	The container type is MPEG Elementary Stream.
XA_CONTAINERTYPE_MPEG_PS	The container type is MPEG Program Stream.

Value	Description
XA_CONTAINERTYPE_MPEG_TS	The container type is MPEG Transport Stream.
XA_CONTAINERTYPE_QT	The container type is QuickTime.
XA_CONTAINERTYPE_WAV	The container type is WAV.
XA_CONTAINERTYPE_XMF_0	The container type is XMF Type 0.
XA_CONTAINERTYPE_XMF_1	The container type is XMF Type 1.
XA_CONTAINERTYPE_XMF_2	The container type is Mobile XMF (XMF Type 2).
XA_CONTAINERTYPE_XMF_3	The container type is Mobile XMF with Audio Clips (XMF Type 3).
XA_CONTAINERTYPE_XMF_GENERIC	The container type is the XMF Meta File Format (no particular XMF File Type)
XA_CONTAINERTYPE_AMR	This container type is the file storage format variant of AMR (the magic number in the header can be used to disambiguate between AMR-NB and AMR-WB).
XA_CONTAINERTYPE_AAC	This container type is for ADIF and ADTS variants of AAC. This refers to AAC in .aac files.
XA_CONTAINERTYPE_3GPP	The container type is 3GPP.
XA_CONTAINERTYPE_3GA	This container type is an audio-only variant of the 3GPP format, mainly used in 3G phones.
XA_CONTAINERTYPE_RM	This container type is Real Media.
XA_CONTAINERTYPE_DMF	This container type is Divx media format.
XA_CONTAINERTYPE_SMF	This container type is a standard MIDI file (SMF) [SP-MIDI].
XA_CONTAINERTYPE_MOBILE_DLS	This container type is a Mobile DLS file [mDLS].
XA_CONTAINERTYPE_OGG	This container type is a OGG.

9.2.24 XA_DATAFORMAT

```
#define XA_DATAFORMAT_MIME       ((XAuint32) 0x00000001)
#define XA_DATAFORMAT_PCM        ((XAuint32) 0x00000002)
#define XA_DATAFORMAT_RAWIMAGE   ((XAuint32) 0x00000003)
#define XA_DATAFORMAT_PCM_EX           ((XAuint32) 0x00000004)
```

These values represent the possible data locators:

Value	Description
XA_DATAFORMAT_MIME	Data format is the specified as a MIME type.
XA_DATAFORMAT_PCM	Data format is PCM. **(Deprecated)**
XA_DATAFORMAT_RAWIMAGE	Data format is raw image.
XA_DATAFORMAT_PCM_EX	Data format is PCM_EX.

9.2.25 XA_DATALOCATOR

```
#define XA_DATALOCATOR_NULL             ((XAuint32) 0x00000000)
#define XA_DATALOCATOR_URI              ((XAuint32) 0x00000001)
#define XA_DATALOCATOR_ADDRESS          ((XAuint32) 0x00000002)
#define XA_DATALOCATOR_IODEVICE         ((XAuint32) 0x00000003)
#define XA_DATALOCATOR_OUTPUTMIX        ((XAuint32) 0x00000004)
#define XA_DATALOCATOR_NATIVEDISPLAY    ((XAuint32) 0x00000005)
#define XA_DATALOCATOR_RESERVED6        ((XAuint32) 0x00000006)
#define XA_DATALOCATOR_RESERVED7        ((XAuint32) 0x00000007)
#define XA_DATALOCATOR_MEDIAOBJECT      ((XAuint32) 0x00000008)
#define XA_DATALOCATOR_CONTENTPIPE      ((XAuint32) 0x00000009)
```

These values represent the possible data locators.

Value	Description
XA_DATALOCATOR_NULL	No data will be generated or consumed.
XA_DATALOCATOR_URI	Data resides at the specified URI.
XA_DATALOCATOR_ADDRESS	Data is stored at the specified memory-mapped address.
XA_DATALOCATOR_IODEVICE	Data will be generated or consumed by the specified IO device. Note: for audio output use the output mix.
XA_DATALOCATOR_OUTPUTMIX	Data will be consumed by the specified audio output mix.
XA_DATALOCATOR_NATIVEDISPLAY	Data will be rendered to the specified native display.
XA_DATALOCATOR_RESERVED6	Reserved value.
XA_DATALOCATOR_RESERVED7	Reserved value.
XA_DATALOCATOR_MEDIAOBJECT	Data will be generated or consumed by a media object.
XA_DATALOCATOR_CONTENTPIPE	Data will be generated or consumed by a content pipe.

9.2.26 XA_DEFAULTDEVICEID

```
#define XA_DEFAULTDEVICEID_AUDIOINPUT  ((XAuint32) 0xFFFFFFFF)
#define XA_DEFAULTDEVICEID_AUDIOOUTPUT ((XAuint32) 0xFFFFFFFE)
#define XA_DEFAULTDEVICEID_LED         ((XAuint32) 0xFFFFFFFD)
#define XA_DEFAULTDEVICEID_VIBRA       ((XAuint32) 0xFFFFFFFC)
#define XA_DEFAULTDEVICEID_CAMERA      ((XAuint32) 0xFFFFFFFB)
```

This macro may be used with any method that manipulates device IDs.

Value	Description
XA_DEFAULTDEVICEID_AUDIOINPUT	Identifier denoting the set of input devices that the implementation receives audio from by default.
XA_DEFAULTDEVICEID_AUDIOOUTPUT	Identifier denoting the set of output devices that the implementation sends audio to by default.
XA_DEFAULTDEVICEID_LED	Identifier denoting default LED array device.
XA_DEFAULTDEVICEID_VIBRA	Identifier denoting default vibra device.

Value	Description
XA_DEFAULTDEVICEID_CAMERA	Identifier denoting default camera device.

9.2.27 XA_DEVICECONNECTION

```
#define XA_DEVCONNECTION_INTEGRATED          ((XAint16) 0x0001)
#define XA_DEVCONNECTION_ATTACHED_WIRED      ((XAint16) 0x0100)
#define XA_DEVCONNECTION_ATTACHED_WIRELESS   ((XAint16) 0x0200)
#define XA_DEVCONNECTION_NETWORK             ((XAint16) 0x0400)
```

These macros list the various types of I/O device connections possible. These connections are mutually exclusive for a given I/O device.

Value	Description
XA_DEVCONNECTION_INTEGRATED	I/O device is integrated onto the system (that is, mobile phone, music player, etc.).
XA_DEVCONNECTION_ATTACHED_WIRED	I/O device is connected to the system via a wired connection. Additional macros might be added if more granularity is needed for each wired connection (such as USB, proprietary, etc.).
XA_DEVCONNECTION_ATTACHED_WIRELESS	I/O device is connected to the system via a wireless connection. Additional macros might be added if more granularity is needed for each wireless connection (such as Bluetooth, etc.).
XA_DEVCONNECTION_NETWORK	I/O device is connected to the system via *some* kind of network connection (either wired or wireless). This is different from the above connections (such as Bluetooth headset or wired accessory) in the sense that this connection could be to a remote device that could be quite distant geographically (unlike a Bluetooth headset or a wired headset that are in close proximity to the system). Also, a network connection implies going through some kind of network routing infrastructure that is not covered by the attached macros above. A Bluetooth headset or a wired headset represents a peer-to-peer connection, whereas a network connection does not. Examples of such network audio I/O devices include remote content servers that feed audio input to the system or a remote media renderer that plays out audio from the system, transmitted to it across a network.

9.2.28 XA_DEVICELOCATION

```
#define XA_DEVLOCATION_HANDSET   ((XAint16) 0x0001)
#define XA_DEVLOCATION_HEADSET   ((XAint16) 0x0002)
#define XA_DEVLOCATION_CARKIT    ((XAint16) 0x0003)
#define XA_DEVLOCATION_DOCK      ((XAint16) 0x0004)
#define XA_DEVLOCATION_REMOTE    ((XAint16) 0x0005)
```

These macros list the location of the I/O device.

Value	Description
XA_DEVLOCATION_HANDSET	I/O device is on the handset.
XA_DEVLOCATION_HEADSET	I/O device is on a headset.
XA_DEVLOCATION_CARKIT	I/O device is on a carkit.
XA_DEVLOCATION_DOCK	I/O device is on a dock.
XA_DEVLOCATION_REMOTE	I/O device is in a remote location, most likely connected via some kind of a network.

Although it might seem like XA_DEVLOCATION_REMOTE is redundant since it is currently used with only XA_DEVCONNECTION_NETWORK, it is needed since none of the other device location macros fit a device whose connection type is XA_DEVCONNECTION_NETWORK.

9.2.29 XA_DEVICESCOPE

```
#define XA_DEVSCOPE_UNKNOWN      ((XAint16) 0x0001)
#define XA_DEVSCOPE_ENVIRONMENT  ((XAint16) 0x0002)
#define XA_DEVSCOPE_USER         ((XAint16) 0x0003)
```

These macros list the scope of the I/O device with respect to the end user. These macros help the application to make routing decisions based on the type of content (such as audio) being rendered. For example, telephony downlink will always default to a “user” audio output device unless specifically changed by the user.

Value	Description
XA_DEVSCOPE_UNKNOWN	I/O device can have either a user scope or an environment scope or an as-yet-undefined scope. Good examples of audio I/O devices with such a scope would be line-in and line-out jacks. It is difficult to tell what types of devices will be plugged into these jacks. I/O devices connected via a network connection also fall into this category.
XA_DEVSCOPE_ENVIRONMENT	I/O device allows environmental (public) input or playback of content (such as audio). For example, an integrated loudspeaker is an “environmental” audio output device, since audio rendered to it can be heard by multiple people. Similarly, a microphone that can accept audio from multiple people is an “environmental” audio input device.
XA_DEVSCOPE_USER	I/O device allows input from or playback of content (such as audio) to a single user. For example, an earpiece speaker is a single-user audio output device since audio rendered to it can be heard only by one person. Similarly, the integrated microphone on a mobile phone is a single-user input device – it accepts input from just one person.

9.2.30 XA_DOMAINTYPE

```
#define XA_DOMAINTYPE_AUDIO             0x00000001
#define XA_DOMAINTYPE_VIDEO             0x00000002
#define XA_DOMAINTYPE_IMAGE             0x00000003
#define XA_DOMAINTYPE_TIMEDTEXT         0x00000004
#define XA_DOMAINTYPE_MIDI              0x00000005
#define XA_DOMAINTYPE_VENDOR            0xFFFFFFFE
#define XA_DOMAINTYPE_UNKNOWN           0xFFFFFFFF
```

These values are used to determine which domain the functionality is associated with.

Value	Description
XA_DOMAINTYPE_AUDIO	Audio domain based functionality.
XA_DOMAINTYPE_VIDEO	Video domain based functionality.
XA_DOMAINTYPE_IMAGE	Imaging domain based functionality.
XA_DOMAINTYPE_TIMEDTEXT	Timed Text domain based functionality.
XA_DOMAINTYPE_MIDI	MIDI domain based functionality.
XA_DOMAINTYPE_VENDOR	Custom domain based functionality. Functionality associated with this domain is implementation specific.
XA_DOMAINTYPE_UNKNOWN	Unknown stream domain. This domain type represents an unrecognizeable stream type.

9.2.31 XA_DYNAMIC_ITF_EVENT

```
#define XA_DYNAMIC_ITF_EVENT_RUNTIME_ERROR \
    ((XAuint32) 0x00000001)
#define XA_DYNAMIC_ITF_EVENT_ASYNC_TERMINATION \
    ((XAuint32) 0x00000002)
#define XA_DYNAMIC_ITF_EVENT_RESOURCES_LOST \
    ((XAuint32) 0x00000003)
#define XA_DYNAMIC_ITF_EVENT_RESOURCES_LOST_PERMANENTLY \
    ((XAuint32) 0x00000004)
#define XA_DYNAMIC_ITF_EVENT_RESOURCES_AVAILABLE \
    ((XAuint32) 0x00000005)
```

These values are used for identifying events used for dynamic interface managerment.

Value	Description
XA_DYNAMIC_ITF_EVENT_RUNTIME_ERROR	Runtime error.
XA_DYNAMIC_ITF_EVENT_ASYNC_TERMINATION	An asynchronous operation has terminated.

Value	Description
XA_DYNAMIC_ITF_EVENT_RESOURCES_LOST	Resources have been stolen from the dynamically managed interface, causing it to become Suspended.
XA_DYNAMIC_ITF_EVENT_RESOURCES_LOST_PERMANENTLY	Resources have been stolen from the dynamically managed interface, causing it to become unrecoverable.
XA_DYNAMIC_ITF_EVENT_RESOURCES_AVAILABLE	Resources have become available, which may enable the dynamically managed interface to resume.

9.2.32 XA_ENGINEOPTION

```
#define XA_ENGINEOPTION_THREADSAFE     ((XAuint32) 0x00000001)
#define XA_ENGINEOPTION_LOSSOFCONTROL  ((XAuint32) 0x00000002)
#define XA_ENGINEOPTION_MAJORVERSION   ((XAuint32) 0x00000003)
#define XA_ENGINEOPTION_MINORVERSION   ((XAuint32) 0x00000004)
#define XA_ENGINEOPTION_STEPVERSION    ((XAuint32) 0x00000005)
```

Engine object creation options (see section 6.1).

Value	Description
XA_ENGINEOPTION_THREADSAFE	Thread safe engine creation option used with XAEngineOption structure (see section 9.1.19). If the data field of the XAEngineOption structure is set to XA_BOOLEAN_TRUE, the engine object is created in thread-safe mode. Otherwise the engine object is created a non-thread-safe mode (see section 4.1.1).
XA_ENGINEOPTION_LOSSOFCONTROL	Global loss-of-control setting used with XAEngineOption structure (see section 9.1.19). If the data field of the XAEngineOption structure is set to XA_BOOLEAN_TRUE, the engine object allows loss-of-control notifications to occur on interfaces. Otherwise, none of the interfaces exhibits loss-of-control behavior. This global setting is best suited for applications that are interested in coarse-grained loss-of-control functionality - either it is allowed for that instance of the engine object or not. See XAObjectItf for details on loss-of-control.
XA_ENGINEOPTION_MAJORVERSION	The API major version for the requested engine object. The data field of the XAEngineOption structure is set to the integer major version of the requested engine object. The default value is 1.
XA_ENGINEOPTION_MINORVERSION	The API minor version for the requested engine object. The data field of the XAEngineOption structure is set to the integer minor version of the requested engine object. The default value is 0.
XA_ENGINEOPTION_STEPVERSION	The API step version for the requested engine object. The data field of the XAEngineOption structure is set to the integer step version of the requested engine object. Because step versions are backwards compatible, a higher step version of the engine than requested may be returned. The default value is 0.

9.2.33 XA_EQUALIZER

```
#define XA_EQUALIZER_UNDEFINED   ((XAuint16) 0xFFFF)
```

This value is used when equalizer setting is not defined.

Value	Description
XA_EQUALIZER_UNDEFINED	The setting is not defined.

9.2.34 XA_FOCUSPOINTS

```
#define XA_FOCUSPOINTS_ONE              ((XAuint32) 0x00000001)
#define XA_FOCUSPOINTS_THREE_3X1        ((XAuint32) 0x00000002)
#define XA_FOCUSPOINTS_FIVE_CROSS       ((XAuint32) 0x00000003)
#define XA_FOCUSPOINTS_SEVEN_CROSS      ((XAuint32) 0x00000004)
#define XA_FOCUSPOINTS_NINE_SQUARE      ((XAuint32) 0x00000005)
#define XA_FOCUSPOINTS_ELEVEN_CROSS     ((XAuint32) 0x00000006)
#define XA_FOCUSPOINTS_TWELVE_3X4       ((XAuint32) 0x00000007)
#define XA_FOCUSPOINTS_TWELVE_4X3       ((XAuint32) 0x00000008)
#define XA_FOCUSPOINTS_SIXTEEN_SQUARE   ((XAuint32) 0x00000009)
#define XA_FOCUSPOINTS_CUSTOM           ((XAuint32) 0x0000000A)
```

These macros are used to describe the camera's focus point pattern. The patterns are used to set the active focus points and to retrieve which points are in focus. The focus points are represented using as bits of a `XAuint32` value. The focus point pattern is the pattern of available focus points particular to a camera. For a given camera with a given pattern, the application may set the active focus by some subset of points within the camera's pattern. Likewise the application may query the subset of points in focus for a given camera with a given pattern. A 32 bit mask represents such a subset of points where each bit position in the mask corresponds to a point in the pattern. Each pattern definition below specifies the point to bit correspondence by labelling each point with the appropriate bit position.

Value	Description
XA_FOCUSPOINTS_ONE	Single focus point positioned in the center.
XA_FOCUSPOINTS_THREE_3X1	Three focus points positioned in a horizontal line.
XA_FOCUSPOINTS_FIVE_CROSS	Five focus points positioned in a cross pattern.
XA_FOCUSPOINTS_SEVEN_CROSS	Seven focus points positioned in a cross pattern.
XA_FOCUSPOINTS_NINE_SQUARE	Nine focus points positioned in a square pattern.
XA_FOCUSPOINTS_ELEVEN_CROSS	Eleven focus points positioned in a cross pattern.
XA_FOCUSPOINTS_TWELVE_3X4	Twelve focus points positioned in a three wide by four tall pattern.
XA_FOCUSPOINTS_TWELVE_4X3	Twelve focus points positioned in a four wide by three tall pattern.
XA_FOCUSPOINTS_SIXTEEN_SQUARE	Sixteen focus points positioned in square pattern.
XA_FOCUSPOINTS_CUSTOM	Custom focus points, allows for a use-selectable pattern.

XA_FOCUSPOINTS_ONE

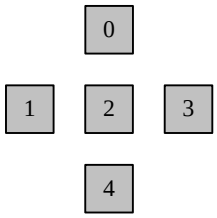
Focus point pattern for XA_FOCUSPOINTS_ONE.

XA_FOCUSPOINTS_THREE_3X1

Focus point pattern for XA_FOCUSPOINTS_THREE_3X1.

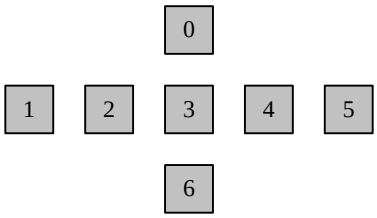
XA_FOCUSPOINTS_FIVE_CROSS

Focus point pattern for XA_FOCUSPOINTS_FIVE_CROSS.



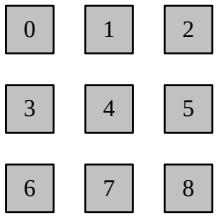
XA_FOCUSPOINTS_SEVEN_CROSS

Focus point pattern for XA_FOCUSPOINTS_SEVEN_CROSS.



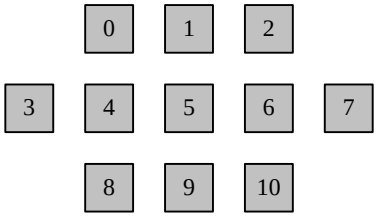
XA_FOCUSPOINTS_NINE_SQUARE

Focus point pattern for XA_FOCUSPOINTS_NINE_SQUARE.



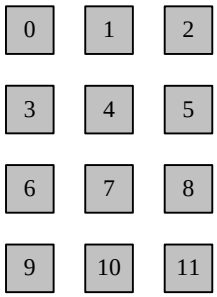
XA_FOCUSPOINTS_ELEVEN_CROSS

Focus point pattern for XA_FOCUSPOINTS_ELEVEN_CROSS.



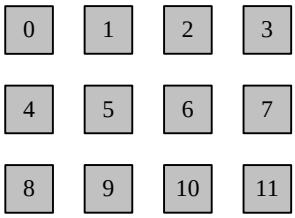
XA_FOCUSPOINTS_TWELVE_3X4

Focus point pattern for XA_FOCUSPOINTS_TWELVE_3X4.



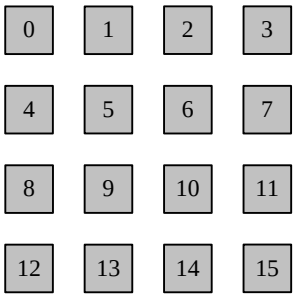
XA_FOCUSPOINTS_TWELVE_4X3

Focus point pattern for XA_FOCUSPOINTS_TWELVE_4X3.



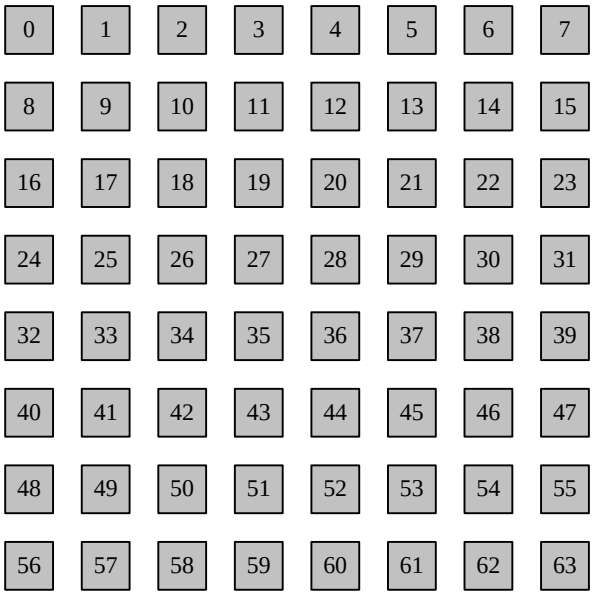
XA_FOCUSPOINTS_SIXTEEN_4X4

Focus point pattern for XA_FOCUSPOINTS_SIXTEEN_4X4.



XA_FOCUSPOINTS_CUSTOM

Focus point pattern for XA_FOCUSPOINTS_CUSTOM.



9.2.35 XA_FREQRANGE

```
#define XA_FREQRANGE_FMEUROAMERICA     ((XAuint8) 0x01)
#define XA_FREQRANGE_FMJAPAN           ((XAuint8) 0x02)
#define XA_FREQRANGE_AMLW              ((XAuint8) 0x03)
#define XA_FREQRANGE_AMMW              ((XAuint8) 0x04)
#define XA_FREQRANGE_AMSW              ((XAuint8) 0x05)
```

These macros are used to specify the frequency range and the modulation.

Value	Description
XA_FREQRANGE_FMEUROAMERICA	European and American FM frequency range.
XA_FREQRANGE_FMJAPAN	Japanese FM frequency range.
XA_FREQRANGE_AMLW	AM Long Wave.
XA_FREQRANGE_AMMW	AM Medium Wave.
XA_FREQRANGE_AMSW	AM Short Wave.

9.2.36 XA_IMAGECODEC

```
#define XA_IMAGECODEC_JPEG ((XAuint32) 0x00000001)
#define XA_IMAGECODEC_GIF  ((XAuint32) 0x00000002)
#define XA_IMAGECODEC_BMP  ((XAuint32) 0x00000003)
#define XA_IMAGECODEC_PNG  ((XAuint32) 0x00000004)
#define XA_IMAGECODEC_TIFF ((XAuint32) 0x00000005)
#define XA_IMAGECODEC_RAW  ((XAuint32) 0x00000006)
```

These macros are used to set the image encoding format.

Value	Description
XA_IMAGECODEC_JPEG	JPEG/JFIF image format.
XA_IMAGECODEC_GIF	GIF (Graphics Interchange Format) image format.
XA_IMAGECODEC_BMP	BMP (Windows Bitmap) image format.
XA_IMAGECODEC_PNG	PNG (Portable Network Graphics) image format.
XA_IMAGECODEC_TIFF	TIFF (Tagged Image File Format) image format.
XA_IMAGECODEC_RAW	RAW image format. Use XA_COLORFORMAT to set color format.

9.2.37 XA_IMAGEEFFECT

```
#define XA_IMAGEEFFECT_MONOCHROME      ((XAuint32) 0x00000001)
#define XA_IMAGEEFFECT_NEGATIVE        ((XAuint32) 0x00000002)
#define XA_IMAGEEFFECT_SEPIA           ((XAuint32) 0x00000003)
#define XA_IMAGEEFFECT_EMBOSS          ((XAuint32) 0x00000004)
#define XA_IMAGEEFFECT_PAINTBRUSH      ((XAuint32) 0x00000005)
#define XA_IMAGEEFFECT_SOLARIZE        ((XAuint32) 0x00000006)
#define XA_IMAGEEFFECT_CARTOON         ((XAuint32) 0x00000007)
```

These macros are used to set the image effect type.

Value	Description
XA_IMAGEEFFECT_MONOCHROME	Monochrome image effect.
XA_IMAGEEFFECT_NEGATIVE	Negative image effect.
XA_IMAGEEFFECT_SEPIA	Sepia image effect.
XA_IMAGEEFFECT_EMBOSS	Emboss image effect.
XA_IMAGEEFFECT_PAINTBRUSH	Paintbrush image effect.
XA_IMAGEEFFECT_SOLARIZE	Solarize image effect.
XA_IMAGEEFFECT_CARTOON	Cartoon image effect.

9.2.38 XA_IODEVICE

```
#define XA_IODEVICE_AUDIOINPUT   ((XAuint32) 0x00000001)
#define XA_IODEVICE_LEDARRAY     ((XAuint32) 0x00000002)
#define XA_IODEVICE_VIBRA        ((XAuint32) 0x00000003)
#define XA_IODEVICE_CAMERA       ((XAuint32) 0x00000004)
#define XA_IODEVICE_RADIO        ((XAuint32) 0x00000005)
#define XA_IODEVICE_AUDIOOUTPUT  ((XAuint32) 0x00000006)
```

These macros are used when creating I/O device data sources and sinks.

Value	Description
XA_IODEVICE_AUDIOINPUT	Device for audio input such as microphone or line-in.
XA_IODEVICE_LEDARRAY	Device for LED arrays.
XA_IODEVICE_VIBRA	Device for vibrators.

Value	Description
XA_IODEVICE_CAMERA	Device for camera used for capturing images.
XA_IODEVICE_RADIO	Device for tuning radio signals.
XA_IODEVICE_AUDIOOUTPUT	Device for audio output.

9.2.39 XA_METADATA_FILTER

```
#define XA_METADATA_FILTER_KEY          ((XAuint8) 0x01)
#define XA_METADATA_FILTER_LANG         ((XAuint8) 0x02)
#define XA_METADATA_FILTER_ENCODING     ((XAuint8) 0x04)
```

Bit-masks for metadata filtering criteria.

Value	Description
XA_METADATA_FILTER_KEY	Enable filtering by key.
XA_METADATA_FILTER_LANG	Enable filtering by language / country code.
XA_METADATA_FILTER_ENCODING	Enable filtering by value encoding.

9.2.40 XA_METADATATRAVERSALMODE

```
#define XA_METADATATRAVERSALMODE_ALL   ((XAuint32) 0x00000001)
#define XA_METADATATRAVERSALMODE_NODE  ((XAuint32) 0x00000002)
```

XA_METADATATRAVERSALMODE represents a method of traversing metadata within a file.

Value	Description
XA_METADATATRAVERSALMODE_ALL	Search the file linearly without considering its logical organization.
XA_METADATATRAVERSALMODE_NODE	Search by individual nodes, boxes, chunks, etc. within a file. (This is the default mode, with the default active node being the root node.)

9.2.41 XA_MIDIBANK

```
#define XA_MIDIBANK_DEVICE      0x00000001
#define XA_MIDIBANK_CUSTOM      0x00000002
```

These values specify the MIDI instrument bank(s) used. It is worth nothing that multiple soundbanks might be used to play content in a single MIDI file.

Value	Description
XA_MIDIBANK_DEVICE	Used to indicate that only the default MIDI instrument bank(s) that are part of the implementation on the device are used.
XA_MIDIBANK_CUSTOM	Used to indicate that custom MIDI instrument bank(s) in addition to or instead of those specified by XA_MIDIBANK_DEVICE are used. Includes Mobile DLS soundbanks. Example: This macro would be used for a MIDI stream from a Mobile XMF file that uses both the default MIDI instrument banks as well as Mobile DLS.

9.2.42 XA_MIDI_UNKNOWN

```
#define XA_MIDI_UNKNOWN         0xFFFFFFFF
```

Value	Description
XA_MIDI_UNKNOWN	Unknown value for MIDI stream attribute.

9.2.43 XA_MILLIBEL

```
#define XA_MILLIBEL_MIN    ((XAmillibel) (-XA_MILLIBEL_MAX-10))
#define XA_MILLIBEL_MAX    ((XAmillibel) 0x7FFF)
```

Limit values for millibel units.

Value	Description
XA_MILLIBEL_MIN	Minimum volume level. This volume may be treated as silence in some implementations.
XA_MILLIBEL_MAX	Maximum volume level.

9.2.44 XA_MILLIHERTZ_MAX

```
#define XA_MILLIHERTZ_MAX  ((XAmilliHertz) 0xFFFFFFFF)
```

Limit value for milliHertz unit.

Value	Description
XA_MILLIHERTZ_MAX	A macro for representing the maximum possible frequency.

9.2.45 XA_MILLIMETER_MAX

```
#define XA_MILLIMETER_MAX  ((XAmillimeter) 0x7FFFFFFF)
```

Limit value for millimeter unit.

Value	Description

Value	Description
XA_MILLIMETER_MAX	A macro for representing the maximum possible positive distance.

9.2.46 XA_NODE_PARENT

```
#define XA_NODE_PARENT     ((XAuint32) 0xFFFFFFFF)
```

XA_NODE_PARENT is used by XAMetadataTraversalItf::SetActiveNode to set the current scope to the node's parent.

Value	Description
XA_NODE_PARENT	Used for setting the active parent node.

9.2.47 XA_NODETYPE

```
#define XA_NODETYPE_UNSPECIFIED  ((XAuint32) 0x00000001)
#define XA_NODETYPE_AUDIO        ((XAuint32) 0x00000002)
#define XA_NODETYPE_VIDEO        ((XAuint32) 0x00000003)
#define XA_NODETYPE_IMAGE        ((XAuint32) 0x00000004)
```

XA_NODETYPE represents the type of a node.

Value	Description
XA_NODETYPE_UNSPECIFIED	Unspecified node type.
XA_NODETYPE_AUDIO	Audio node.
XA_NODETYPE_VIDEO	Video node.
XA_NODETYPE_IMAGE	Image node.

9.2.48 XA_OBJECT_EVENT

```
#define XA_OBJECT_EVENT_RUNTIME_ERROR              ((XAuint32) 0x00000001)
#define XA_OBJECT_EVENT_ASYNC_TERMINATION          ((XAuint32) 0x00000002)
#define XA_OBJECT_EVENT_RESOURCES_LOST             ((XAuint32) 0x00000003)
#define XA_OBJECT_EVENT_RESOURCES_AVAILABLE        ((XAuint32) 0x00000004)
#define XA_OBJECT_EVENT_ITF_CONTROL_TAKEN          ((XAuint32) 0x00000005)
#define XA_OBJECT_EVENT_ITF_CONTROL_RETURNED       ((XAuint32) 0x00000006)
#define XA_OBJECT_EVENT_ITF_PARAMETERS_CHANGED     ((XAuint32) 0x00000007)
```

The macros identify the various event notifiations that an object may emit.

Value	Description
XA_OBJECT_EVENT_RUNTIME_ERROR	Runtime error.
XA_OBJECT_EVENT_ASYNC_TERMINATION	An asynchronous operation has terminated.
XA_OBJECT_EVENT_RESOURCES_LOST	Resources have been stolen from the object, causing it to become Unrealized or Suspended.
XA_OBJECT_EVENT_RESOURCES_AVAILABLE	Resources have become available, which may enable the object to recover.
XA_OBJECT_EVENT_ITF_CONTROL_TAKEN	An interface has lost control. This event cannot be followed by another XA_OBJECT_EVENT_ITF_CONTROL_TAKEN event (for the interface in question).
XA_OBJECT_EVENT_ITF_CONTROL_RETURNED	Control was returned to an interface. This event cannot be followed by another XA_OBJECT_EVENT_ITF_CONTROL_RETURNED event (for the interface in question).
XA_OBJECT_EVENT_ITF_PARAMETERS_CHANGED	Some of the parameters of the interface in question were changed by other entity. (If the application wants to know the new values, it should use getters.) This event can only occur (for the interface in question) between XA_OBJECT_EVENT_ITF_CONTROL_TAKEN and XA_OBJECT_EVENT_ITF_CONTROL_RETURNED events.

9.2.49 XA_OBJECT_STATE

```
#define XA_OBJECT_STATE_UNREALIZED     ((XAuint32) 0x00000001)
#define XA_OBJECT_STATE_REALIZED       ((XAuint32) 0x00000002)
#define XA_OBJECT_STATE_SUSPENDED      ((XAuint32) 0x00000003)
```

These macros are used to identify the object states.

Value	Description
XA_OBJECT_STATE_UNREALIZED	Unrealized state.
XA_OBJECT_STATE_REALIZED	Realized state.

Value	Description
XA_OBJECT_STATE_SUSPENDED	Suspended state.

9.2.50 XA_OBJECTID

```
#define XA_OBJECTID_ENGINE             ((XAuint32) 0x00000001)
#define XA_OBJECTID_LEDDEVICE          ((XAuint32) 0x00000002)
#define XA_OBJECTID_VIBRADEVICE        ((XAuint32) 0x00000003)
#define XA_OBJECTID_MEDIAPLAYER        ((XAuint32) 0x00000004)
#define XA_OBJECTID_MEDIARECORDER      ((XAuint32) 0x00000005)
#define XA_OBJECTID_RADIODEVICE        ((XAuint32) 0x00000006)
#define XA_OBJECTID_OUTPUTMIX          ((XAuint32) 0x00000007)
#define XA_OBJECTID_METADATAEXTRACTOR  ((XAuint32) 0x00000008)
#define XA_OBJECTID_CAMERADEVICE       ((XAuint32) 0x00000009)
```

These macros are the object identifiers use while querying for the supported interfaces

Value	Description
XA_OBJECTID_ENGINE	Engine Object ID.
XA_OBJECTID_LEDDEVICE	LED Device Object ID.
XA_OBJECTID_VIBRADEVICE	Vibra Device Object ID.
XA_OBJECTID_MEDIAPLAYER	Media Player Object ID.
XA_OBJECTID_MEDIARECORDER	Media Recorder Object ID.
XA_OBJECTID_RADIODEVICE	Radio Device Object ID.
XA_OBJECTID_OUTPUTMIX	Output Mix Object ID.
XA_OBJECTID_METADATAEXTRACTOR	Metadata Extractor Object ID.
XA_OBJECTID_CAMERADEVICE	Camera Device Object ID.

9.2.51 XA_ORIENTATION

```
#define XA_ORIENTATION_UNKNOWN   ((XAuint32) 0x00000001)
#define XA_ORIENTATION_OUTWARDS  ((XAuint32) 0x00000002)
#define XA_ORIENTATION_INWARDS   ((XAuint32) 0x00000003)
```

These macros are used to describe the device orientation relative to the user.

Value	Description
XA_ORIENTATION_UNKNOWN	The pointing direction of the device is user configurable or cannot be known for some other reason.
XA_ORIENTATION_INWARDS	The device is pointing towards the user when the device is held in a natural way.
XA_ORIENTATION_OUTWARDS	The device is pointing away from the user when the device is held in a natural way.

9.2.52 XA_PCM_REPRESENTATION

```
#define XA_PCM_REPRESENTATION_SIGNED_INT        ((XAuint32) 0x00000001)
#define XA_PCM_REPRESENTATION_UNSIGNED_INT      ((XAuint32) 0x00000002)
#define XA_PCM_REPRESENTATION_FLOAT             ((XAuint32) 0x00000003)
```

XA_PCM_REPRESENTATION denotes the type of PCM data.

Value	Description
XA_PCM_REPRESENTATION_SIGNED_INT	Signed integer data.
XA_PCM_REPRESENTATION_UNSIGNED_INT	Unsigned integer data.
XA_PCM_REPRESENTATION_FLOAT	Floating-point data.

9.2.53 XA_PCMSAMPLEFORMAT

```
#define XA_PCMSAMPLEFORMAT_FIXED_8     ((XAuint16) 0x0008)
#define XA_PCMSAMPLEFORMAT_FIXED_16    ((XAuint16) 0x0010)
#define XA_PCMSAMPLEFORMAT_FIXED_20    ((XAuint16) 0x0014)
#define XA_PCMSAMPLEFORMAT_FIXED_24    ((XAuint16) 0x0018)
#define XA_PCMSAMPLEFORMAT_FIXED_28    ((XAuint16) 0x001C)
#define XA_PCMSAMPLEFORMAT_FIXED_32    ((XAuint16) 0x0020)
#define XA_PCMSAMPLEFORMAT_FIXED_64    ((XAuint16) 0x0040)
```

These macros list the various sample formats that are possible on audio input and output devices.

Value	Description
XA_PCMSAMPLEFORMAT_FIXED_8	Fixed-point 8-bit samples in 8-bit container.
XA_PCMSAMPLEFORMAT_FIXED_16	Fixed-point 16-bit samples in 16 bit container.
XA_PCMSAMPLEFORMAT_FIXED_20	Fixed-point 20-bit samples in 32 bit container left-justifed.
XA_PCMSAMPLEFORMAT_FIXED_24	Fixed-point 24-bit samples in 32 bit container left-justifed.
XA_PCMSAMPLEFORMAT_FIXED_28	Fixed-point 28-bit samples in 32 bit container left-justifed.
XA_PCMSAMPLEFORMAT_FIXED_32	Fixed-point 32-bit samples in 32 bit container left-justifed.
XA_PCMSAMPLEFORMAT_FIXED_64	Fixed-point 64-bit samples in 64 bit container left-justifed.

9.2.54 XA_PLAYEVENT

```
#define XA_PLAYEVENT_HEADATEND          ((XAuint32) 0x00000001)
#define XA_PLAYEVENT_HEADATMARKER       ((XAuint32) 0x00000002)
#define XA_PLAYEVENT_HEADATNEWPOS       ((XAuint32) 0x00000004)
#define XA_PLAYEVENT_HEADMOVING         ((XAuint32) 0x00000008)
#define XA_PLAYEVENT_HEADSTALLED        ((XAuint32) 0x00000010)
#define XA_PLAYEVENT_DURATIONUPDATED    ((XAuint32) 0x00000020)
```

These values represent the possible play events.

Value	Description
XA_PLAYEVENT_HEADATEND	Playback head is at the end of the current content and the player has paused.
XA_PLAYEVENT_HEADATMARKER	Playback head is at the specified marker position.
XA_PLAYEVENT_HEADATNEWPOS	Playback head is at a new position (period between notifications is specified in by application).
XA_PLAYEVENT_HEADMOVING	Playback head has begun to move.
XA_PLAYEVENT_HEADSTALLED	Playback head has temporarily stopped moving.
XA_PLAYEVENT_DURATIONUPDATED	The duration of the content has been updated.

9.2.55 XA_PLAYSTATE

```
#define XA_PLAYSTATE_STOPPED     ((XAuint32) 0x00000001)
#define XA_PLAYSTATE_PAUSED      ((XAuint32) 0x00000002)
#define XA_PLAYSTATE_PLAYING     ((XAuint32) 0x00000003)
```

These values represent the playback state of an object

Value	Description
XA_PLAYSTATE_STOPPED	Player is stopped. The playback head is forced to the beginning of the content and is not trying to move.
XA_PLAYSTATE_PAUSED	Player is paused. The playback head may be anywhere within the content but is not trying to move.
XA_PLAYSTATE_PLAYING	Player is playing. The playback head may be anywhere within the content and is trying to move.

9.2.56 XA_PREFETCHEVENT

```
#define XA_PREFETCHEVENT_STATUSCHANGE        ((XAuint32) 0x00000001)
#define XA_PREFETCHEVENT_FILLLEVELCHANGE     ((XAuint32) 0x00000002)
```

These values represent the possible prefetch related events.

Value	Description
XA_PREFETCHEVENT_STATUSCHANGE	Prefetch status has changed.
XA_PREFETCHEVENT_FILLLEVELCHANGE	Prefetch fill level has changed.

9.2.57 XA_PREFETCHSTATUS

```
#define XA_PREFETCHSTATUS_UNDERFLOW          ((XAuint32) 0x00000001)
#define XA_PREFETCHSTATUS_SUFFICIENTDATA     ((XAuint32) 0x00000002)
#define XA_PREFETCHSTATUS_OVERFLOW           ((XAuint32) 0x00000003)
```

These values represent the possible status of a player's prefetching operation.

Value	Description
XA_PREFETCHSTATUS_UNDERFLOW	Playback is suffering due to data starvation.
XA_PREFETCHSTATUS_SUFFICIENTDATA	Playback is not suffering due to data starvation or spillover.
XA_PREFETCHSTATUS_OVERFLOW	Playback is suffering due to data spillover.

9.2.58 XA_PRIORITY

```
#define XA_PRIORITY_LOWEST       ((XAuint32) 0xFFFFFFFF)
#define XA_PRIORITY_VERYLOW      ((XAuint32) 0xE0000000)
#define XA_PRIORITY_LOW          ((XAuint32) 0xC0000000)
#define XA_PRIORITY_BELOWNORMAL  ((XAuint32) 0xA0000000)
#define XA_PRIORITY_NORMAL       ((XAuint32) 0x7FFFFFFF)
#define XA_PRIORITY_ABOVENORMAL  ((XAuint32) 0x60000000)
#define XA_PRIORITY_HIGH         ((XAuint32) 0x40000000)
#define XA_PRIORITY_VERYHIGH     ((XAuint32) 0x20000000)
#define XA_PRIORITY_HIGHEST      ((XAuint32) 0x00000000)
```

Convenient macros representing various different priority levels, for use with the `SetPriority` method.

Value	Description
XA_PRIORITY_LOWEST	The lowest specifiable priority.
XA_PRIORITY_VERYLOW	Very low priority.
XA_PRIORITY_LOW	Low priority.
XA_PRIORITY_BELOWNORMAL	Below normal priority.
XA_PRIORITY_NORMAL	Normal priority given to objects.
XA_PRIORITY_ABOVENORMAL	Above normal priority.
XA_PRIORITY_HIGH	High priority.
XA_PRIORITY_VERYHIGH	Very high priority.
XA_PRIORITY_HIGHEST	Highest specifiable priority.

9.2.59 XA_PROFILE

```
#define XA_PROFILES_MEDIA_PLAYER             ((XAint16) 0x0001)
#define XA_PROFILES_MEDIA_PLAYER_RECORDER    ((XAint16) 0x0002)
#define XA_PROFILES_PLUS_MIDI                ((XAint16) 0x0004)
```

Macros used to report profiles supported. Valid combinations are `XA_PROFILES_MEDIA_PLAYER`, `(XA_PROFILES_MEDIA_PLAYER | XA_PROFILES_MEDIA_PLAYER_RECORDER)`, `(XA_PROFILES_MEDIA_PLAYER | XA_PROFILES_PLUS_MIDI)` and

(XA_PROFILES_MEDIA_PLAYER | XA_PROFILES_MEDIA_PLAYER_RECORDER | XA_PROFILES_PLUS_MIDI).

Value	Description
XA_PROFILES_MEDIA_PLAYER	Media player profile. For a description of the profile, see section 2.3.
XA_PROFILES_MEDIA_PLAYER_RECORDER	Media player/recorder profile. For a description of the profile, see section 2.3.
XA_PROFILES_PLUS_MIDI	"+ MIDI" designation. For a description of the designation, see section 2.5. XA_PROFILES_PLUS_MIDI cannot be set alone, it must be set along with XA_PROFILE_MEDIA_PLAYER or XA_PROFILE_MEDIA_PLAYER_RECORDER.

9.2.60 XA_RADIO_EVENT

```
#define XA_RADIO_EVENT_ANTENNA_STATUS_CHANGED      ((XAuint32) 0x00000001)
#define XA_RADIO_EVENT_FREQUENCY_CHANGED           ((XAuint32) 0x00000002)
#define XA_RADIO_EVENT_FREQUENCY_RANGE_CHANGED     ((XAuint32) 0x00000003)
#define XA_RADIO_EVENT_PRESET_CHANGED              ((XAuint32) 0x00000004)
#define XA_RADIO_EVENT_SEEK_COMPLETED              ((XAuint32) 0x00000005)
#define XA_RADIO_EVENT_STEREO_STATUS_CHANGED       ((XAuint32) 0x00000006)
#define XA_RADIO_EVENT_SIGNAL_STRENGTH_CHANGED     ((XAuint32) 0x00000007)
#define XA_RADIO_EVENT_SQUELCH_CHANGED             ((XAuint32) 0x00000008)
#define XA_RADIO_EVENT_FREQUENCY_ERROR             ((XAuint32) 0x00000009)
#define XA_RADIO_EVENT_FREQUENCY_RANGE_ERROR       ((XAuint32) 0x0000000A)
```

These macros are used to define the radio related event and the event specific parameters used by `xaRadioCallback()`.

Value	Description
XA_RADIO_EVENT_ANTENNA_STATUS_CHANGED	This event indicates that the status of the antenna was changed. (Some devices contain antennas that can be unplugged from the device.) The `eventIntData` parameter for this event is not used and shall be ignored. The `eventBooleanData` parameter for this event is `XA_BOOLEAN_FALSE` if the antenna was detached and `XA_BOOLEAN_TRUE` if the antenna was attached.
XA_RADIO_EVENT_FREQUENCY_CHANGED	This event indicates that the frequency was changed. This can be caused either by manual tuning with `SetFrequency` or by automatic switching based on RDS. The `eventIntData` parameter for this event contains the new frequency in Hertz. The `eventBooleanData` parameter for this event is `XA_BOOLEAN_TRUE` if the the change of the frequency was automatic and caused by RDS related reason and `XA_BOOLEAN_FALSE` otherwise.
XA_RADIO_EVENT_FREQUENCY_RANGE_CHANGED	This event indicates that the frequency range was changed. The `eventIntData` parameter for this event contains the new frequency range. See `XA_Freq_Range` macros. The `eventBooleanData` parameter for this event is not used and shall be ignored.
XA_RADIO_EVENT_PRESET_CHANGED	This event indicates that a preset has been modified. This can be caused also by other applications that change the presets. The `eventIntData` parameter for this event contains the index of the preset that was modified. The `eventBooleanData` parameter for this event is not used and shall be ignored.

Value	Description
XA_RADIO_EVENT_SEEK_COMPLETED	This event indicates that the seek is completed and the frequency of the tuner is the one that is given as eventIntData parameter, or if nothing was found, the starting frequency (that will be then given as parameter). The eventBooleanData parameter for this event is not used and shall be ignored.
XA_RADIO_EVENT_STEREO_STATUS_CHANGED	This event indicates that the signal received by the radio has changed from stereo to mono or from mono to stereo. The eventIntData parameter for this event is not used and shall be ignored. The eventBooleanData parameter for this event is XA_BOOLEAN_FALSE if the signal changed from stereo to mono and XA_BOOLEAN_TRUE if the signal changed from mono to stereo.
XA_RADIO_EVENT_SIGNAL_STRENGTH_CHANGED	This event indicates that the strength of the signal received by the radio has changed. The eventIntData parameter for this event contains the new signal strength in percent. The eventBooleanData parameter for this event is not used and shall be ignored.
XA_RADIO_EVENT_SQUELCH_CHANGED	This event indicates that the squelch has been modified by another application. The eventIntData parameter for this event is not used and shall be ignored. The eventBooleanData parameter for this event is not used and shall be ignored.
XA_RADIO_EVENT_FREQUENCY_ERROR	This event indicates that an error has occurred while changing the frequency via the SetFrequency method. The eventIntData parameter for this event contains one of the following two error codes, XA_RESULT_FEATURE_UNSUPPORTED and XA_RESULT_UNKNOWN_ERROR. The eventBooleanData parameter for this event is is not used and shall be ignored.
XA_RADIO_EVENT_FREQUENCY_RANGE_ERROR	This event indicates that an error has occurred while changing the frequency range via the SetFrequencyRange method. The eventIntData parameter for this event contains one of the following two error codes, XA_RESULT_FEATURE_UNSUPPORTED and XA_RESULT_UNKNOWN_ERROR. The eventBooleanData parameter for this event is not used and shall be ignored.

9.2.61 XA_RATECONTROLMODE

```
#define XA_RATECONTROLMODE_CONSTANTBITRATE   ((XAuint32) 0x00000001)
#define XA_RATECONTROLMODE_VARIABLEBITRATE   ((XAuint32) 0x00000002)
```

These macros are used to set the rate control mode.

Value	Description
XA_RATECONTROLMODE_CONSTANTBITRATE	Constant bitrate mode.
XA_RATECONTROLMODE_VARIABLEBITRATE	Variable bitrate mode.

9.2.62 XA_RATEPROP

```
#define XA_RATEPROP_STAGGEREDVIDEO     ((XAuint32) 0x00000001)
#define XA_RATEPROP_SMOOTHVIDEO        ((XAuint32) 0x00000002)
#define XA_RATEPROP_SILENTAUDIO        ((XAuint32) 0x00000100)
#define XA_RATEPROP_STAGGEREDAUDIO     ((XAuint32) 0x00000200)
#define XA_RATEPROP_NOPITCHCORAUDIO    ((XAuint32) 0x00000400)
#define XA_RATEPROP_PITCHCORAUDIO      ((XAuint32) 0x00000800)
```

These values represent the rate-related properties of an object.

Value	Description
XA_RATEPROP_STAGGEREDVIDEO	Displays staggered video. Implementation presents a subset of video frames (dropping some intermediate frames). This property accommodates limitations of rewind and high speed fast forward.
XA_RATEPROP_SMOOTHVIDEO	Displays smooth video. Implementation presents all video frames, but the presentation timing respects the current rate.
XA_RATEPROP_SILENTAUDIO	Silences audio output. This property accommodates limitations of rewind and high speed fast-forward.
XA_RATEPROP_STAGGEREDAUDIO	Plays small chunks of audio at 1x forward, skipping segments of audio between chunks. The progression of the playback head between chunks obeys the direction and speed implied by the current rate. This property accommodates limitations of rewind and high speed fast forward.
XA_RATEPROP_NOPITCHCORAUDIO	Plays audio at the current rate, but without pitch correction.
XA_RATEPROP_PITCHCORAUDIO	Plays audio at the current rate, but with pitch correction.

9.2.63 XA_RDS_EVENT

```
#define XA_RDS_EVENT_NEW_PI      ((XAuint16) 0x0001)
#define XA_RDS_EVENT_NEW_PTY     ((XAuint16) 0x0002)
#define XA_RDS_EVENT_NEW_PS      ((XAuint16) 0x0004)
#define XA_RDS_EVENT_NEW_RT      ((XAuint16) 0x0008)
#define XA_RDS_EVENT_NEW_RT_PLUS ((XAuint16) 0x0010)
#define XA_RDS_EVENT_NEW_CT      ((XAuint16) 0x0020)
#define XA_RDS_EVENT_NEW_TA      ((XAuint16) 0x0040)
#define XA_RDS_EVENT_NEW_TP      ((XAuint16) 0x0080)
#define XA_RDS_EVENT_NEW_ALARM   ((XAuint16) 0x0100)
```

These macros are used to define which of the RDS fields have changed.

Value	Description
XA_RDS_NEW_PI	The Programme Identification code has changed. The eventData parameter for this event is not used and shall be ignored.
XA_RDS_EVENT_NEW_PTY	The Programme TYpe has changed. The eventData parameter for this event is not used and shall be ignored.
XA_RDS_EVENT_NEW_PS	The Programme Service name has changed. The eventData parameter for this event is not used and shall be ignored.
XA_RDS_EVENT_NEW_RT	The Radio Text has changed. The eventData parameter for this event is not used and shall be ignored.
XA_RDS_EVENT_NEW_RT_PLUS	A Radio Text plus information element has changed. The RT+ class code of the changed information element is given as eventData parameter of the callback. This event is posted also in the case when an information element is cleared. Then Only one event is send in cases when there is an additional descriptor element associated with another information element.
XA_RDS_EVENT_NEW_CT	The Clock Time and date has changed. The eventData parameter for this event is not used and shall be ignored.
XA_RDS_EVENT_NEW_TA	The Traffic Announcement has changed. The eventData parameter for this event is not used and shall be ignored.
XA_RDS_EVENT_NEW_TP	The Traffic Programme has changed. The eventData parameter for this event is not used and shall be ignored.
XA_RDS_EVENT_NEW_ALARM	The Alarm status has changed. The eventData parameter for this event is not used and shall be ignored.

9.2.64 XA_RDSPROGRAMMETYPE

The interpretation of values of this type depends on the origin of the RDS broadcast: in North America, a slightly different standard, RBDS, is used. These PTY codes are defined by static values XA_RDSPROGRAMMETYPE_RBDSPTY_XXX, for example XA_RDSPROGRAMMETYPE_RBDSTYPE_SOFTROCK. Elsewhere, including Europe, the RDS standard is used. In these areas, the PTY codes are defined by static values XA_PROGRAMMETYPE_RDSTYPE_XXX, for example XA_PROGRAMMETYPE_RDSPTY_CHILDRENSPROGRAMMES.

```
#define XA_RDSPROGRAMMETYPE_RDSPTY_NONE \
    ((XAuint32) 0x00000000)
#define XA_RDSPROGRAMMETYPE_RDSPTY_NEWS \
    ((XAuint32) 0x00000001)
#define XA_RDSPROGRAMMETYPE_RDSPTY_CURRENTAFFAIRS \
    ((XAuint32) 0x00000002)
#define XA_RDSPROGRAMMETYPE_RDSPTY_INFORMATION \
    ((XAuint32) 0x00000003)
#define XA_RDSPROGRAMMETYPE_RDSPTY_SPORT \
    ((XAuint32) 0x00000004)
#define XA_RDSPROGRAMMETYPE_RDSPTY_EDUCATION \
    ((XAuint32) 0x00000005)
#define XA_RDSPROGRAMMETYPE_RDSPTY_DRAMA \
    ((XAuint32) 0x00000006)
#define XA_RDSPROGRAMMETYPE_RDSPTY_CULTURE \
    ((XAuint32) 0x00000007)
#define XA_RDSPROGRAMMETYPE_RDSPTY_SCIENCE \
    ((XAuint32) 0x00000008)
#define XA_RDSPROGRAMMETYPE_RDSPTY_VARIEDSPEECH \
    ((XAuint32) 0x00000009)
#define XA_RDSPROGRAMMETYPE_RDSPTY_POPMUSIC \
    ((XAuint32) 0x0000000A)
#define XA_RDSPROGRAMMETYPE_RDSPTY_ROCKMUSIC \
    ((XAuint32) 0x0000000B)
#define XA_RDSPROGRAMMETYPE_RDSPTY_EASYLISTENING \
    ((XAuint32) 0x0000000C)
#define XA_RDSPROGRAMMETYPE_RDSPTY_LIGHTCLASSICAL \
    ((XAuint32) 0x0000000D)
#define XA_RDSPROGRAMMETYPE_RDSPTY_SERIOUSCLASSICAL \
    ((XAuint32) 0x0000000E)
#define XA_RDSPROGRAMMETYPE_RDSPTY_OTHERMUSIC \
    ((XAuint32) 0x0000000F)
#define XA_RDSPROGRAMMETYPE_RDSPTY_WEATHER \
    ((XAuint32) 0x00000010)
#define XA_RDSPROGRAMMETYPE_RDSPTY_FINANCE \
    ((XAuint32) 0x00000011)
#define XA_RDSPROGRAMMETYPE_RDSPTY_CHILDRENSPROGRAMMES \
    ((XAuint32) 0x00000012)
#define XA_RDSPROGRAMMETYPE_RDSPTY_SOCIALAFFAIRS \
    ((XAuint32) 0x00000013)
#define XA_RDSPROGRAMMETYPE_RDSPTY_RELIGION \
    ((XAuint32) 0x00000014)
#define XA_RDSPROGRAMMETYPE_RDSPTY_PHONEIN \
    ((XAuint32) 0x00000015)
#define XA_RDSPROGRAMMETYPE_RDSPTY_TRAVEL \
```

```
    ((XAuint32) 0x00000016)
#define XA_RDSPROGRAMMETYPE_RDSPTY_LEISURE \
    ((XAuint32) 0x00000017)
#define XA_RDSPROGRAMMETYPE_RDSPTY_JAZZMUSIC \
    ((XAuint32) 0x00000018)
#define XA_RDSPROGRAMMETYPE_RDSPTY_COUNTRYMUSIC \
    ((XAuint32) 0x00000019)
#define XA_RDSPROGRAMMETYPE_RDSPTY_NATIONALMUSIC \
    ((XAuint32) 0x0000001A)
#define XA_RDSPROGRAMMETYPE_RDSPTY_OLDIESMUSIC \
    ((XAuint32) 0x0000001B)
#define XA_RDSPROGRAMMETYPE_RDSPTY_FOLKMUSIC \
    ((XAuint32) 0x0000001C)
#define XA_RDSPROGRAMMETYPE_RDSPTY_DOCUMENTARY \
    ((XAuint32) 0x0000001D)
#define XA_RDSPROGRAMMETYPE_RDSPTY_ALARMTEST \
    ((XAuint32) 0x0000001E)
#define XA_RDSPROGRAMMETYPE_RDSPTY_ALARM \
    ((XAuint32) 0x0000001F)

#define XA_RDSPROGRAMMETYPE_RBDSPTY_NONE \
    ((XAuint32) 0x00000000)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_NEWS \
    ((XAuint32) 0x00000001)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_INFORMATION \
    ((XAuint32) 0x00000002)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_SPORTS \
    ((XAuint32) 0x00000003)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_TALK \
    ((XAuint32) 0x00000004)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_ROCK \
    ((XAuint32) 0x00000005)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_CLASSICROCK \
    ((XAuint32) 0x00000006)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_ADULTHITS \
    ((XAuint32) 0x00000007)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_SOFTROCK \
    ((XAuint32) 0x00000008)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_TOP40 \
    ((XAuint32) 0x00000009)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_COUNTRY \
    ((XAuint32) 0x0000000A)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_OLDIES \
    ((XAuint32) 0x0000000B)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_SOFT \
    ((XAuint32) 0x0000000C)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_NOSTALGIA \
    ((XAuint32) 0x0000000D)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_JAZZ \
    ((XAuint32) 0x0000000E)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_CLASSICAL \
    ((XAuint32) 0x0000000F)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_RHYTHMANDBLUES \
    ((XAuint32) 0x00000010)
```

```
#define XA_RDSPROGRAMMETYPE_RBDSPTY_SOFTRHYTHMANDBLUES \
    ((XAuint32) 0x00000011)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_LANGUAGE \
    ((XAuint32) 0x00000012)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_RELIGIOUSMUSIC \
    ((XAuint32) 0x00000013)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_RELIGIOUSTALK \
    ((XAuint32) 0x00000014)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_PERSONALITY \
    ((XAuint32) 0x00000015)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_PUBLIC \
    ((XAuint32) 0x00000016)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_COLLEGE \
    ((XAuint32) 0x00000017)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED1 \
    ((XAuint32) 0x00000018)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED2 \
    ((XAuint32) 0x00000019)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED3 \
    ((XAuint32) 0x0000001A)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED4 \
    ((XAuint32) 0x0000001B)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED5 \
    ((XAuint32) 0x0000001C)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_WEATHER \
    ((XAuint32) 0x0000001D)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_EMERGENCYTEST \
    ((XAuint32) 0x0000001E)
#define XA_RDSPROGRAMMETYPE_RBDSPTY_EMERGENCY \
    ((XAuint32) 0x0000001F)
```

The RDS Programme Types are:

Value	Description
XA_RDSPROGRAMMETYPE_RDSPTY_NONE	No programme type or undefined.
XA_RDSPROGRAMMETYPE_RDSPTY_NEWS	News.
XA_RDSPROGRAMMETYPE_RDSPTY_CURRENTAFFAIRS	Current Affairs.
XA_RDSPROGRAMMETYPE_RDSPTY_INFORMATION	Information.
XA_RDSPROGRAMMETYPE_RDSPTY_SPORT	Sport.
XA_RDSPROGRAMMETYPE_RDSPTY_EDUCATION	Education.
XA_RDSPROGRAMMETYPE_RDSPTY_DRAMA	Drama.
XA_RDSPROGRAMMETYPE_RDSPTY_CULTURE	Culture.
XA_RDSPROGRAMMETYPE_RDSPTY_SCIENCE	Science.
XA_RDSPROGRAMMETYPE_RDSPTY_VARIEDSPEECH	Varied.
XA_RDSPROGRAMMETYPE_RDSPTY_POPMUSIC	Pop Music.
XA_RDSPROGRAMMETYPE_RDSPTY_ROCKMUSIC	Rock Music.
XA_RDSPROGRAMMETYPE_RDSPTY_EASYLISTENING	Easy Listening.

Value	Description
XA_RDSPROGRAMMETYPE_RDSPTY_LIGHTCLASSICAL	Light Classical.
XA_RDSPROGRAMMETYPE_RDSPTY_SERIOUSCLASSICAL	Serious Classical.
XA_RDSPROGRAMMETYPE_RDSPTY_OTHERMUSIC	Other Music.
XA_RDSPROGRAMMETYPE_RDSPTY_WEATHER	Weather.
XA_RDSPROGRAMMETYPE_RDSPTY_FINANCE	Finance.
XA_RDSPROGRAMMETYPE_RDSPTY_CHILDRENSPROGRAMMES	Children's Programmes.
XA_RDSPROGRAMMETYPE_RDSPTY_SOCIALAFFAIRS	Social Affairs.
XA_RDSPROGRAMMETYPE_RDSPTY_RELIGION	Religion.
XA_RDSPROGRAMMETYPE_RDSPTY_PHONEIN	Phone In.
XA_RDSPROGRAMMETYPE_RDSPTY_TRAVEL	Travel.
XA_RDSPROGRAMMETYPE_RDSPTY_LEISURE	Leisure.
XA_RDSPROGRAMMETYPE_RDSPTY_JAZZMUSIC	Jazz Music.
XA_RDSPROGRAMMETYPE_RDSPTY_COUNTRYMUSIC	Country Music.
XA_RDSPROGRAMMETYPE_RDSPTY_NATIONALMUSIC	National Music.
XA_RDSPROGRAMMETYPE_RDSPTY_OLDIESMUSIC	Oldies Music.
XA_RDSPROGRAMMETYPE_RDSPTY_FOLKMUSIC	Folk Music.
XA_RDSPROGRAMMETYPE_RDSPTY_DOCUMENTARY	Documentary.
XA_RDSPROGRAMMETYPE_RDSPTY_ALARMTEST	Alarm Test.
XA_RDSPROGRAMMETYPE_RDSPTY_ALARM	Alarm.

The RBDS Programme Types are:

Value	Description
XA_RDSPROGRAMMETYPE_RBDSPTY_NONE	No programme type or undefined.
XA_RDSPROGRAMMETYPE_RBDSPTY_NEWS	News.
XA_RDSPROGRAMMETYPE_RBDSPTY_INFORMATION	Information.
XA_RDSPROGRAMMETYPE_RBDSPTY_SPORTS	Sports.
XA_RDSPROGRAMMETYPE_RBDSPTY_TALK	Talk.
XA_RDSPROGRAMMETYPE_RBDSPTY_ROCK	Rock.
XA_RDSPROGRAMMETYPE_RBDSPTY_CLASSICROCK	Classic Rock.
XA_RDSPROGRAMMETYPE_RBDSPTY_ADULTHITS	Adult Hits.
XA_RDSPROGRAMMETYPE_RBDSPTY_SOFTROCK	Soft Rock.
XA_RDSPROGRAMMETYPE_RBDSPTY_TOP40	Top 40.
XA_RDSPROGRAMMETYPE_RBDSPTY_COUNTRY	Country.
XA_RDSPROGRAMMETYPE_RBDSPTY_OLDIES	Oldies.

Value	Description
XA_RDSPROGRAMMETYPE_RBDSPTY_SOFT	Soft.
XA_RDSPROGRAMMETYPE_RBDSPTY_NOSTALGIA	Nostalgia.
XA_RDSPROGRAMMETYPE_RBDSPTY_JAZZ	Jazz.
XA_RDSPROGRAMMETYPE_RBDSPTY_CLASSICAL	Classical.
XA_RDSPROGRAMMETYPE_RBDSPTY_RHYTHMANDBLUES	Rhythm and Blues.
XA_RDSPROGRAMMETYPE_RBDSPTY_SOFTRHYTHMANDBLUES	Soft Rhythm and Blues.
XA_RDSPROGRAMMETYPE_RBDSPTY_LANGUAGE	Language.
XA_RDSPROGRAMMETYPE_RBDSPTY_RELIGIOUSMUSIC	Religious Music.
XA_RDSPROGRAMMETYPE_RBDSPTY_RELIGIOUSTALK	Religious Talk.
XA_RDSPROGRAMMETYPE_RBDSPTY_PERSONALITY	Personality.
XA_RDSPROGRAMMETYPE_RBDSPTY_PUBLIC	Public.
XA_RDSPROGRAMMETYPE_RBDSPTY_COLLEGE	College.
XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED1	Unassigned.
XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED2	Unassigned.
XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED3	Unassigned.
XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED4	Unassigned.
XA_RDSPROGRAMMETYPE_RBDSPTY_UNASSIGNED5	Unassigned.
XA_RDSPROGRAMMETYPE_RBDSPTY_WEATHER	Weather.
XA_RDSPROGRAMMETYPE_RBDSPTY_EMERGENCYTEST	Emergency Test.
XA_RDSPROGRAMMETYPE_RBDSPTY_EMERGENCY	Emergency.

9.2.65 XA_RDSRTPLUS

```
#define XA_RDSRTPLUS_ITEMTITLE              ((XAuint8) 0x01)
#define XA_RDSRTPLUS_ITEMALBUM              ((XAuint8) 0x02)
#define XA_RDSRTPLUS_ITEMTRACKNUMBER        ((XAuint8) 0x03)
#define XA_RDSRTPLUS_ITEMARTIST             ((XAuint8) 0x04)
#define XA_RDSRTPLUS_ITEMCOMPOSITION        ((XAuint8) 0x05)
#define XA_RDSRTPLUS_ITEMMOVEMENT           ((XAuint8) 0x06)
#define XA_RDSRTPLUS_ITEMCONDUCTOR          ((XAuint8) 0x07)
#define XA_RDSRTPLUS_ITEMCOMPOSER           ((XAuint8) 0x08)
#define XA_RDSRTPLUS_ITEMBAND               ((XAuint8) 0x09)
#define XA_RDSRTPLUS_ITEMCOMMENT            ((XAuint8) 0x0A)
#define XA_RDSRTPLUS_ITEMGENRE              ((XAuint8) 0x0B)
#define XA_RDSRTPLUS_INFONEWS               ((XAuint8) 0x0C)
#define XA_RDSRTPLUS_INFONEWSLOCAL          ((XAuint8) 0x0D)
#define XA_RDSRTPLUS_INFOSTOCKMARKET        ((XAuint8) 0x0E)
#define XA_RDSRTPLUS_INFOSPORT              ((XAuint8) 0x0F)
#define XA_RDSRTPLUS_INFOLOTTERY            ((XAuint8) 0x10)
#define XA_RDSRTPLUS_INFOHOROSCOPE          ((XAuint8) 0x11)
#define XA_RDSRTPLUS_INFODAILYDIVERSION     ((XAuint8) 0x12)
#define XA_RDSRTPLUS_INFOHEALTH             ((XAuint8) 0x13)
#define XA_RDSRTPLUS_INFOEVENT              ((XAuint8) 0x14)
#define XA_RDSRTPLUS_INFOSZENE              ((XAuint8) 0x15)
#define XA_RDSRTPLUS_INFOCINEMA             ((XAuint8) 0x16)
#define XA_RDSRTPLUS_INFOTV                 ((XAuint8) 0x17)
#define XA_RDSRTPLUS_INFODATETIME           ((XAuint8) 0x18)
#define XA_RDSRTPLUS_INFOWEATHER            ((XAuint8) 0x19)
#define XA_RDSRTPLUS_INFOTRAFFIC            ((XAuint8) 0x1A)
#define XA_RDSRTPLUS_INFOALARM              ((XAuint8) 0x1B)
#define XA_RDSRTPLUS_INFOADVISERTISEMENT    ((XAuint8) 0x1C)
#define XA_RDSRTPLUS_INFOURL                ((XAuint8) 0x1D)
#define XA_RDSRTPLUS_INFOOTHER              ((XAuint8) 0x1E)
#define XA_RDSRTPLUS_STATIONNAMESHORT       ((XAuint8) 0x1F)
#define XA_RDSRTPLUS_STATIONNAMELONG        ((XAuint8) 0x20)
#define XA_RDSRTPLUS_PROGRAMNOW             ((XAuint8) 0x21)
#define XA_RDSRTPLUS_PROGRAMNEXT            ((XAuint8) 0x22)
#define XA_RDSRTPLUS_PROGRAMPART            ((XAuint8) 0x23)
#define XA_RDSRTPLUS_PROGRAMHOST            ((XAuint8) 0x24)
#define XA_RDSRTPLUS_PROFRAMEDITORIALSTAFF  ((XAuint8) 0x25)
#define XA_RDSRTPLUS_PROGRAMFREQUENCY       ((XAuint8) 0x26)
#define XA_RDSRTPLUS_PROGRAMHOMEPAGE        ((XAuint8) 0x27)
#define XA_RDSRTPLUS_PROGRAMSUBCHANNEL      ((XAuint8) 0x28)
#define XA_RDSRTPLUS_PHONEHOTLINE           ((XAuint8) 0x29)
#define XA_RDSRTPLUS_PHONESTUDIO            ((XAuint8) 0x2A)
#define XA_RDSRTPLUS_PHONEOTHER             ((XAuint8) 0x2B)
#define XA_RDSRTPLUS_SMSSTUDIO              ((XAuint8) 0x2C)
#define XA_RDSRTPLUS_SMSOTHER               ((XAuint8) 0x2D)
#define XA_RDSRTPLUS_EMAILHOTLINE           ((XAuint8) 0x2E)
#define XA_RDSRTPLUS_EMAILSTUDIO            ((XAuint8) 0x2F)
#define XA_RDSRTPLUS_EMAILOTHER             ((XAuint8) 0x30)
#define XA_RDSRTPLUS_MMSOTHER               ((XAuint8) 0x31)
#define XA_RDSRTPLUS_CHAT                   ((XAuint8) 0x32)
#define XA_RDSRTPLUS_CHATCENTER             ((XAuint8) 0x33)
#define XA_RDSRTPLUS_VOTEQUESTION           ((XAuint8) 0x34)
#define XA_RDSRTPLUS_VOTECENTER             ((XAuint8) 0x35)
#define XA_RDSRTPLUS_OPENCLASS45            ((XAuint8) 0x36)
```

```
#define XA_RDSRTPLUS_OPENCLASS55                ((XAuint8) 0x37)
#define XA_RDSRTPLUS_OPENCLASS56                ((XAuint8) 0x38)
#define XA_RDSRTPLUS_OPENCLASS57                ((XAuint8) 0x39)
#define XA_RDSRTPLUS_OPENCLASS58                ((XAuint8) 0x3A)
#define XA_RDSRTPLUS_PLACE                      ((XAuint8) 0x3B)
#define XA_RDSRTPLUS_APPOINTMENT                ((XAuint8) 0x3C)
#define XA_RDSRTPLUS_IDENTIFIER                 ((XAuint8) 0x3D)
#define XA_RDSRTPLUS_PURCHASE                   ((XAuint8) 0x3E)
#define XA_RDSRTPLUS_GETDATA                    ((XAuint8) 0x3F)
```

The macros are sued to specify Class codes for RT+ content types and they are defined in Radiotext plus (RT+) Specification Version 2.1. See RDS Forum 2006-07-21 - R06/040_1.

Category	enum	Code	RT+ Classes
ITEM	XA_RDSRTPLUS_ITEMTITLE	1	TITLE
	XA_RDSRTPLUS_ITEMALBUM	2	ALBUM
	XA_RDSRTPLUS_ITEMTRACKUMBER	3	TRACKNUMBER
	XA_RDSRTPLUS_ITEMARTIST	4	ARTIST
	XA_RDSRTPLUS_ITEMCOMPOSITION	5	COMPOSITION
	XA_RDSRTPLUS_ITEMMOVEMENT	6	MOVEMENT
	XA_RDSRTPLUS_ITEMCONDUCTOR	7	CONDUCTOR
	XA_RDSRTPLUS_ITEMCOMPOSER	8	COMPOSER
	XA_RDSRTPLUS_ITEMBAND	9	BAND
	XA_RDSRTPLUS_ITEMCOMMENT	10	COMMENT
	XA_RDSRTPLUS_ITEMGENRE	11	GENRE
Info	XA_RDSRTPLUS_INFONEWS	12	NEWS
	XA_RDSRTPLUS_INFONEWSLOCAL	13	NEWS.LOCAL
	XA_RDSRTPLUS_INFOSTOCKMARKET	14	STOCKMARKET
	XA_RDSRTPLUS_INFOSPORT	15	SPORT
	XA_RDSRTPLUS_INFOLOTTERY	16	LOTTERY
	XA_RDSRTPLUS_INFOHOROSCOPE	17	HOROSCOPE
	XA_RDSRTPLUS_INFODAILYDIVERSION	18	DAILY_DIVERSION
	XA_RDSRTPLUS_INFOHEALTH	19	HEALTH
	XA_RDSRTPLUS_INFOEVENT	20	EVENT
	XA_RDSRTPLUS_INFOSZENE	21	SZENE
	XA_RDSRTPLUS_INFOCINEMA	22	CINEMA
	XA_RDSRTPLUS_INFOTV	23	TV
	XA_RDSRTPLUS_INFODATETIME	24	DATE_TIME
	XA_RDSRTPLUS_INFOWEATHER	25	WEATHER
	XA_RDSRTPLUS_INFOTRAFFIC	26	TRAFFIC

Category	enum	Code	RT+ Classes
	XA_RDSRTPLUS_INFOALARM	27	ALARM
	XA_RDSRTPLUS_INFOADVISERTISEMENT	28	ADVERTISEMENT
	XA_RDSRTPLUS_INFOURL	29	URL
	XA_RDSRTPLUS_INFOOTHER	30	OTHER
Programme	XA_RDSRTPLUS_STATIONNAMESHORT	31	STATIONNAME.SHORT
	XA_RDSRTPLUS_STATIONNAMELONG	32	STATIONNAME.LONG
	XA_RDSRTPLUS_PROGRAMNOW	33	NOW
	XA_RDSRTPLUS_PROGRAMNEXT	34	NEXT
	XA_RDSRTPLUS_PROGRAMPART	35	PART
	XA_RDSRTPLUS_PROGRAMHOST	36	HOST
	XA_RDSRTPLUS_PROGRAMEDITORIALSTAFF	37	EDITORIAL_STAFF
	XA_RDSRTPLUS_PROGRAMFREQUENCY	38	FREQUENCY
	XA_RDSRTPLUS_PROGRAMHOMEPAGE	39	HOMEPAGE
	XA_RDSRTPLUS_PROGRAMSUBCHANNEL	40	SUBCHANNEL
Interactivity	XA_RDSRTPLUS_PHONEHOTLINE	41	PHONE.HOTLINE
	XA_RDSRTPLUS_PHONESTUDIO	42	PHONE.STUDIO
	XA_RDSRTPLUS_PHONEOTHER	43	PHONE.OTHER
	XA_RDSRTPLUS_SMSSTUDIO	44	SMS.STUDIO
	XA_RDSRTPLUS_SMSOTHER	45	SMS.OTHER
	XA_RDSRTPLUS_EMAILHOTLINE	46	EMAIL.HOTLINE
	XA_RDSRTPLUS_EMAILSTUDIO	47	EMAIL.STUDIO
	XA_RDSRTPLUS_EMAILOTHER	48	EMAIL.OTHER
	XA_RDSRTPLUS_MMSOTHER	49	MMS.OTHER
	XA_RDSRTPLUS_CHAT	50	CHAT
	XA_RDSRTPLUS_CHATCENTER	51	CHAT.CENTER
	XA_RDSRTPLUS_VOTEQUESTION	52	VOTE.QUESTION
	XA_RDSRTPLUS_VOTECENTER	53	VOTE.CENTER
rfu		54	Reserved for Future Use
		55	Reserved for Future Use
Private classes		56	Private classes may be defined by the service provider
		57	Private classes may be defined by the service provider
		58	Private classes may be defined by the service provider

Category	enum	Code	RT+ Classes
Descriptor	XA_RDSRTPLUS_PLACE	59	PLACE
	XA_RDSRTPLUS_APPOINTMENT	60	APPOINTMENT
	XA_RDSRTPLUS_IDENTIFIER	61	IDENTIFIER
	XA_RDSRTPLUS_PURCHASE	62	PURCHASE
	XA_RDSRTPLUS_GETDATA	63	GET_DATA

9.2.66 XA_RECORDEVENT

```
#define XA_RECORDEVENT_HEADATLIMIT      ((XAuint32) 0x00000001)
#define XA_RECORDEVENT_HEADATMARKER     ((XAuint32) 0x00000002)
#define XA_RECORDEVENT_HEADATNEWPOS     ((XAuint32) 0x00000004)
#define XA_RECORDEVENT_HEADMOVING       ((XAuint32) 0x00000008)
#define XA_RECORDEVENT_HEADSTALLED      ((XAuint32) 0x00000010)
#define XA_RECORDEVENT_BUFFER_FULL      ((XAuint32) 0x00000020)
```

These values represent the possible record events.

Value	Description
XA_RECORDEVENT_HEADATLIMIT	Recording head is at the specified duration limit and the recorder has stopped.
XA_RECORDEVENT_HEADATMARKER	Recording head is at the specified marker position.
XA_RECORDEVENT_HEADATNEWPOS	Recording head is at a new position. (Period between notifications is specified by application.)
XA_RECORDEVENT_HEADMOVING	Recording head has begun to move.
XA_RECORDEVENT_HEADSTALLED	Recording head has temporarily stopped moving.
XA_RECORDEVENT_BUFFER_FULL	Recording has reached the end of the memory buffer (i.e. XADataLocator_Address). When the recorder is unable to write any more data (e.g. when the memory buffer it is writing to is full) the recorder transitions to the XA_RECORDSTATE_STOPPED state. This event will not be posted when recording to a file.

9.2.67 XA_RECORDSTATE

```
#define XA_RECORDSTATE_STOPPED   ((XAuint32) 0x00000001)
#define XA_RECORDSTATE_PAUSED    ((XAuint32) 0x00000002)
#define XA_RECORDSTATE_RECORDING ((XAuint32) 0x00000003)
```

These values represent the recording state of an object.

Value	Description
XA_RECORDSTATE_STOPPED	Recorder is stopped. The destination is closed
XA_RECORDSTATE_PAUSED	Recorder is stopped. The destination is open but not receiving captured content.
XA_RECORDSTATE_RECORDING	Recorder is recording. The destination is open and receiving captured content.

9.2.68 XA_RENDERINGHINT

```
#define XA_RENDERINGHINT_NONE ((XAuint32) 0x00000000)
#define XA_RENDERINGHINT_ANTIALIASING ((XAuint32) 0x00000001)
```

These values represent rendering hints for image and video processing. They can be used with XAVideoPostProcessingItf.

Value	Description
XA_RENDERINGHINT_NONE	No specific hint is given. The application prefers speed.
XA_RENDERINGHINT_ANTIALIASING	A hint to use anti-aliasing in processing. The application prefers quality.

9.2.69 XA_RESULT

```
#define XA_RESULT_SUCCESS                    ((XAuint32) 0x00000000)
#define XA_RESULT_PRECONDITIONS_VIOLATED     ((XAuint32) 0x00000001)
#define XA_RESULT_PARAMETER_INVALID          ((XAuint32) 0x00000002)
#define XA_RESULT_MEMORY_FAILURE             ((XAuint32) 0x00000003)
#define XA_RESULT_RESOURCE_ERROR             ((XAuint32) 0x00000004)
#define XA_RESULT_RESOURCE_LOST              ((XAuint32) 0x00000005)
#define XA_RESULT_IO_ERROR                   ((XAuint32) 0x00000006)
#define XA_RESULT_BUFFER_INSUFFICIENT        ((XAuint32) 0x00000007)
#define XA_RESULT_CONTENT_CORRUPTED          ((XAuint32) 0x00000008)
#define XA_RESULT_CONTENT_UNSUPPORTED        ((XAuint32) 0x00000009)
#define XA_RESULT_CONTENT_NOT_FOUND          ((XAuint32) 0x0000000A)
#define XA_RESULT_PERMISSION_DENIED          ((XAuint32) 0x0000000B)
#define XA_RESULT_FEATURE_UNSUPPORTED        ((XAuint32) 0x0000000C)
#define XA_RESULT_INTERNAL_ERROR             ((XAuint32) 0x0000000D)
#define XA_RESULT_UNKNOWN_ERROR              ((XAuint32) 0x0000000E)
#define XA_RESULT_OPERATION_ABORTED          ((XAuint32) 0x0000000F)
#define XA_RESULT_CONTROL_LOST               ((XAuint32) 0x00000010)
#define XA_RESULT_READONLY                   ((XAuint32) 0x00000011)
#define XA_RESULT_ENGINEOPTION_UNSUPPORTED   ((XAuint32) 0x00000012)
#define XA_RESULT_SOURCE_SINK_INCOMPATIBLE   ((XAuint32) 0x00000013)
```

The XA_RESULT values are described.

Value	Description
XA_RESULT_SUCCESS	Success.
XA_RESULT_PRECONDITIONS_VIOLATED	Use of the method violates a pre-condition (not including invalid parameters). The pre-conditions are defined in the method specifications.
XA_RESULT_PARAMETER_INVALID	An invalid parameter has been detected. In case of parameters passed by pointer (such as the self-parameters) – if the pointer is corrupt, an implementation's behavior is undefined. However, it is recommended that implementations at least check for NULL-pointers.
XA_RESULT_MEMORY_FAILURE	The method was unable to allocate or release memory.
XA_RESULT_RESOURCE_ERROR	Operation failed due to a lack of resources (usually a result of object realization).

Value	Description
XA_RESULT_RESOURCE_LOST	Operation ignored, since object is in Unrealized or Suspended state.
XA_RESULT_IO_ERROR	Failure due to an I/O error (file or other I/O device).
XA_RESULT_BUFFER_INSUFFICIENT	One or more of the buffers passed to the method is too small to service the request.
XA_RESULT_CONTENT_CORRUPTED	Failure due to corrupted content (also applies for malformed MIDI messages sent programmatically).
XA_RESULT_CONTENT_UNSUPPORTED	Failure due to an unsupported content format (such as unsupported codec).
XA_RESULT_CONTENT_NOT_FOUND	Failed to retrieve content (for example, file not found).
XA_RESULT_PERMISSION_DENIED	Failure due to violation of DRM, user permissions, policies, etc.
XA_RESULT_FEATURE_UNSUPPORTED	Failure due to an unsupported feature. This can occur either when calling `GetInterface()` on an unsupported interface or when attempting to call a method not supported in an interface. This can also occur when an interface with an unknown ID is used (either during object creation or in a `GetInterface()` call). See Section 2.7.
XA_RESULT_INTERNAL_ERROR	Failure due to an (unrecoverable) internal error.
XA_RESULT_UNKNOWN_ERROR	Catch-all error, including system errors. Should never be returned when any of the above errors apply.
XA_RESULT_OPERATION_ABORTED	Operation was aborted as a result of a user request.
XA_RESULT_CONTROL_LOST	Another entity is now controlling the interface and it cannot be controlled by this application currently. `xaObjectCallback` can be used for monitoring this behavior: this error code can only occur between `XA_OBJECT_EVENT_ITF_CONTROL_TAKEN` and `XA_OBJECT_EVENT_ITF_CONTROL_RETURNED` events.
XA_RESULT_READONLY	Operation failed because the parameter in question is Read-Only.
XA_RESULT_ENGINEOPTION_UNSUPPORTED	Failure due to an unsupported engine option.
XA_RESULT_SOURCE_SINK_INCOMPATIBLE	The source and sink being connected are incompatible (incompatibility could be due to format and/or locator type mismatch).

9.2.70 XA_ROOT_NODE_ID

```
#define XA_ROOT_NODE_ID    ((XAint32) 0x7FFFFFFF)
```

This define is used to refer to the root node of the metadata tree.

Value	Description
XA_ROOT_NODE_ID	ID of the root node.

9.2.71 XA_SAMPLINGRATE

```
#define XA_SAMPLINGRATE_8        ((XAuint32) 8000000)
#define XA_SAMPLINGRATE_11_025   ((XAuint32) 11025000)
#define XA_SAMPLINGRATE_12       ((XAuint32) 12000000)
#define XA_SAMPLINGRATE_16       ((XAuint32) 16000000)
#define XA_SAMPLINGRATE_22_05    ((XAuint32) 22050000)
#define XA_SAMPLINGRATE_24       ((XAuint32) 24000000)
#define XA_SAMPLINGRATE_32       ((XAuint32) 32000000)
#define XA_SAMPLINGRATE_44_1     ((XAuint32) 44100000)
#define XA_SAMPLINGRATE_48       ((XAuint32) 48000000)
#define XA_SAMPLINGRATE_64       ((XAuint32) 64000000)
#define XA_SAMPLINGRATE_88_2     ((XAuint32) 88200000)
#define XA_SAMPLINGRATE_96       ((XAuint32) 96000000)
#define XA_SAMPLINGRATE_192      ((XAuint32) 192000000)
```

These macros specify the commonly used sampling rates (in milliHertz) supported by most audio I/O devices.

Value	Description
XA_SAMPLINGRATE_8	8 kHz sampling rate.
XA_SAMPLINGRATE_11_025	11.025 kHz sampling rate.
XA_SAMPLINGRATE_12	12 kHz sampling rate.
XA_SAMPLINGRATE_16	16 kHz sampling rate.
XA_SAMPLINGRATE_22_05	22.05 kHz sampling rate.
XA_SAMPLINGRATE_24	24 kHz sampling rate.
XA_SAMPLINGRATE_32	32 kHz sampling rate.
XA_SAMPLINGRATE_44_1	44.1 kHz sampling rate.
XA_SAMPLINGRATE_48	48 kHz sampling rate.
XA_SAMPLINGRATE_64	64 kHz sampling rate.
XA_SAMPLINGRATE_88_2	88.2 kHz sampling rate.
XA_SAMPLINGRATE_96	96 kHz sampling rate.
XA_SAMPLINGRATE_192	192 kHz sampling rate.

9.2.72 XA_SEEKMODE

```
#define XA_SEEKMODE_FAST         ((XAuint32) 0x0001)
#define XA_SEEKMODE_ACCURATE     ((XAuint32) 0x0002)
```

These values represent seek modes.

The nature of encoded content and of the API implementation may imply tradeoffs between the accuracy and speed of a seek operation. Seek modes afford the application a means to specify which characteristic, accuracy or speed, should be preferred.

For some encoded data formats, dependencies exist between discrete samples/frames. Seeking an exact position may thus imply some latency while the implementation builds up data at the desired position from data preceding it (or

even following it). For example, when the position specified corresponds to a B-frame, the implementation will need to reconstruct the forward and backwards reference frames for that B-frame.

Alternatively, an implementation may seek the independent sample/frame (such as an Intraframe) nearest to the desired position. Although this reduces latency, this approach implies a larger potential distance between the desired position and the position realized by the implementation.

Value	Description
XA_SEEKMODE_FAST	Prefer the speed of a seek over the accuracy of a seek. Upon a `SetPosition()` call, the implementation minimizes latency potentially at the expense of accuracy; effective playback head position may vary slightly from the requested position
XA_SEEKMODE_ACCURATE	Prefer the accuracy of a seek over the speed of a seek. Upon a `SetPosition()` call, the implementation minimizes the distance between the effective playback head position and the requested position, potentially at the price of higher latency.

9.2.73 XA_STEREO_MODE

```
#define XA_STEREOMODE_MONO      ((XAuint32) 0x00000001)
#define XA_STEREOMODE_STEREO    ((XAuint32) 0x00000002)
#define XA_STEREOMODE_AUTO      ((XAuint32) 0x00000003)
```

These macros are used to define the stereo modes.

Value	Description
XA_STEREOMODE_MONO	Forces monaural mode.
XA_STEREOMODE_STEREO	Forces stereo mode.
XA_STEREOMODE_AUTO	Automatic stereo mode. This mode uses the best available mode.

9.2.74 XA_SPEAKER

```
#define XA_SPEAKER_FRONT_LEFT               ((XAuint32) 0x00000001)
#define XA_SPEAKER_FRONT_RIGHT              ((XAuint32) 0x00000002)
#define XA_SPEAKER_FRONT_CENTER             ((XAuint32) 0x00000004)
#define XA_SPEAKER_LOW_FREQUENCY            ((XAuint32) 0x00000008)
#define XA_SPEAKER_BACK_LEFT                ((XAuint32) 0x00000010)
#define XA_SPEAKER_BACK_RIGHT               ((XAuint32) 0x00000020)
#define XA_SPEAKER_FRONT_LEFT_OF_CENTER     ((XAuint32) 0x00000040)
#define XA_SPEAKER_FRONT_RIGHT_OF_CENTER    ((XAuint32) 0x00000080)
#define XA_SPEAKER_BACK_CENTER              ((XAuint32) 0x00000100)
#define XA_SPEAKER_SIDE_LEFT                ((XAuint32) 0x00000200)
#define XA_SPEAKER_SIDE_RIGHT               ((XAuint32) 0x00000400)
#define XA_SPEAKER_TOP_CENTER               ((XAuint32) 0x00000800)
#define XA_SPEAKER_TOP_FRONT_LEFT           ((XAuint32) 0x00001000)
#define XA_SPEAKER_TOP_FRONT_CENTER         ((XAuint32) 0x00002000)
#define XA_SPEAKER_TOP_FRONT_RIGHT          ((XAuint32) 0x00004000)
#define XA_SPEAKER_TOP_BACK_LEFT            ((XAuint32) 0x00008000)
#define XA_SPEAKER_TOP_BACK_CENTER          ((XAuint32) 0x00010000)
#define XA_SPEAKER_TOP_BACK_RIGHT           ((XAuint32) 0x00020000)
```

Speaker location macros used when specifying a channel mask.

Value	Description
XA_SPEAKER_FRONT_LEFT	Front left speaker channel.
XA_SPEAKER_FRONT_RIGHT	Front right speaker channel.
XA_SPEAKER_FRONT_CENTER	Front center speaker channel.
XA_SPEAKER_LOW_FREQUENCY	Low frequency effects (LFE) speaker channel.
XA_SPEAKER_BACK_LEFT	Rear left speaker channel.
XA_SPEAKER_BACK_RIGHT	Rear right speaker channel.
XA_SPEAKER_FRONT_LEFT_OF_CENTER	Front left-of-center speaker channel.
XA_SPEAKER_FRONT_RIGHT_OF_CENTER	Front right-of-center speaker channel.
XA_SPEAKER_BACK_CENTER	Rear center speaker channel.
XA_SPEAKER_SIDE_LEFT	Side left speaker channel.
XA_SPEAKER_SIDE_RIGHT	Side right speaker channel.
XA_SPEAKER_TOP_CENTER	Top center speaker channel.
XA_SPEAKER_TOP_FRONT_LEFT	Top front left speaker channel.
XA_SPEAKER_TOP_FRONT_CENTER	Top front center speaker channel.
XA_SPEAKER_TOP_FRONT_RIGHT	Top front right speaker channel.
XA_SPEAKER_TOP_BACK_LEFT	Top rear left speaker channel.
XA_SPEAKER_TOP_BACK_CENTER	Top rear center speaker channel.
XA_SPEAKER_TOP_BACK_RIGHT	Top rear right speaker channel.

9.2.75 XA_STREAMCBEVENT

```
#define XA_STREAMCBEVENT_PROPERTYCHANGE          ((XAuint32) 0x00000001)
```

These values are used to identify the callback event type.

Value	Description
XA_STREAMCBEVENT_PROPERTYCHANGE	This event indicates that stream property change has occurred. The `streamIndex` parameter identifies the stream with the property change. The `pEventData` parameter for this event is not used and shall be ignored.

9.2.76 XA_TIME

```
#define XA_TIME_UNKNOWN    ((XAuint32) 0xFFFFFFFF)
```

These values are reserved for special designations of playback time that cannot be represented using the normal numeric range.

Value	Description
XA_TIME_UNKNOWN	The duration of playback is unknown (such as the content is a broadcast stream)

9.2.77 XA_VIDEOCODEC

```
#define XA_VIDEOCODEC_MPEG2      ((XAuint32) 0x00000001)
#define XA_VIDEOCODEC_H263       ((XAuint32) 0x00000002)
#define XA_VIDEOCODEC_MPEG4      ((XAuint32) 0x00000003)
#define XA_VIDEOCODEC_AVC        ((XAuint32) 0x00000004)
#define XA_VIDEOCODEC_VC1        ((XAuint32) 0x00000005)
#define XA_VIDEOCODEC_VP8        ((XAuint32) 0x00000006)
```

These macros are used to set the video encoding format.

Value	Description
XA_VIDEOCODEC_MPEG2	MPEG2, also known as H.262 video format.
XA_VIDEOCODEC_H263	ITU H.263 video format.
XA_VIDEOCODEC_MPEG4	MPEG4 video format.
XA_VIDEOCODEC_AVC	MPEG4 Part 10 Advanced Video Coding, also known as H.264 video format.
XA_VIDEOCODEC_VC1	Windows Media Codec video format.
XA_VIDEOCODEC_VP8	VP8 video format

9.2.78 XA_VIDEOMIRROR

```
#define XA_VIDEOMIRROR_NONE            ((XAuint32) 0x00000001)
#define XA_VIDEOMIRROR_VERTICAL        ((XAuint32) 0x00000002)
#define XA_VIDEOMIRROR_HORIZONTAL      ((XAuint32) 0x00000003)
#define XA_VIDEOMIRROR_BOTH            ((XAuint32) 0x00000004)
```

These macros are used to set the video mirroring. They are intended to be used with XAVideoPostProcessingItf.

Value	Description
XA_VIDEOMIRROR_NONE	No mirroring.
XA_VIDEOMIRROR_VERTICAL	Flips the image across the horizontal axis. That is, the topmost parts of the source image will become the lowermost parts of the target image and vice versa.
XA_VIDEOMIRROR_HORIZONTAL	Flips the image across the vertical axis. That is, the leftmost parts of the source image will become the rightmost parts of the target image and vice versa.
XA_VIDEOMIRROR_BOTH	Flips the image across both axes. This equals 180 degrees rotation.

9.2.79 XA_VIDEOPROFILE and XA_VIDEOLEVEL

MPEG-2 Profiles and Levels

```
#define XA_VIDEOPROFILE_MPEG2_SIMPLE   ((XAuint32) 0x00000001)
#define XA_VIDEOPROFILE_MPEG2_MAIN     ((XAuint32) 0x00000002)
#define XA_VIDEOPROFILE_MPEG2_422      ((XAuint32) 0x00000003)
#define XA_VIDEOPROFILE_MPEG2_SNR      ((XAuint32) 0x00000004)
#define XA_VIDEOPROFILE_MPEG2_SPATIAL  ((XAuint32) 0x00000005)
#define XA_VIDEOPROFILE_MPEG2_HIGH     ((XAuint32) 0x00000006)

#define XA_VIDEOLEVEL_MPEG2_LL   ((XAuint32) 0x00000001)
#define XA_VIDEOLEVEL_MPEG2_ML   ((XAuint32) 0x00000002)
#define XA_VIDEOLEVEL_MPEG2_H14  ((XAuint32) 0x00000003)
#define XA_VIDEOLEVEL_MPEG2_HL   ((XAuint32) 0x00000004)
```

These macros are used for defining MPEG-2 video profiles and levels.

Value	Description
XA_VIDEOPROFILE_MPEG2_SIMPLE	MPEG-2 Simple Profile.
XA_VIDEOPROFILE_MPEG2_MAIN	MPEG-2 Main Profile.
XA_VIDEOPROFILE_MPEG2_422	MPEG-2 4:2:2 Profile.
XA_VIDEOPROFILE_MPEG2_SNR	MPEG-2 SNR Profile.
XA_VIDEOPROFILE_MPEG2_SPATIAL	MPEG-2 Spatial Profile.
XA_VIDEOPROFILE_MPEG2_HIGH	MPEG-2 High Profile.
XA_VIDEOLEVEL_MPEG2_LL	MPEG-2 Level Low.
XA_VIDEOLEVEL_MPEG2_ML	MPEG-2 Level Main.
XA_VIDEOLEVEL_MPEG2_H14	MPEG-2 Level High 1440.
XA_VIDEOLEVEL_MPEG2_HL	MPEG-2 Level High.

H.263 Profiles and Levels

```
#define XA_VIDEOPROFILE_H263_BASELINE             ((XAuint32) 0x00000001)
#define XA_VIDEOPROFILE_H263_H320CODING           ((XAuint32) 0x00000002)
#define XA_VIDEOPROFILE_H263_BACKWARDCOMPATIBLE   ((XAuint32) 0x00000003)
#define XA_VIDEOPROFILE_H263_ISWV2                ((XAuint32) 0x00000004)
#define XA_VIDEOPROFILE_H263_ISWV3                ((XAuint32) 0x00000005)
#define XA_VIDEOPROFILE_H263_HIGHCOMPRESSION      ((XAuint32) 0x00000006)
#define XA_VIDEOPROFILE_H263_INTERNET             ((XAuint32) 0x00000007)
#define XA_VIDEOPROFILE_H263_INTERLACE            ((XAuint32) 0x00000008)
#define XA_VIDEOPROFILE_H263_HIGHLATENCY          ((XAuint32) 0x00000009)
```

```
#define XA_VIDEOLEVEL_H263_10    ((XAuint32) 0x00000001)
#define XA_VIDEOLEVEL_H263_20    ((XAuint32) 0x00000002)
#define XA_VIDEOLEVEL_H263_30    ((XAuint32) 0x00000003)
#define XA_VIDEOLEVEL_H263_40    ((XAuint32) 0x00000004)
#define XA_VIDEOLEVEL_H263_45    ((XAuint32) 0x00000005)
#define XA_VIDEOLEVEL_H263_50    ((XAuint32) 0x00000006)
#define XA_VIDEOLEVEL_H263_60    ((XAuint32) 0x00000007)
#define XA_VIDEOLEVEL_H263_70    ((XAuint32) 0x00000008)
```

These macros are used for defining H.263 video profiles and levels.

Value	Description
XA_VIDEOPROFILE_H263_BASELINE	H.263 Baseline Profile.
XA_VIDEOPROFILE_H263_H320CODING	H.263 H.320 Coding Efficiency Version 2 Backward-Compatibility Profile.
XA_VIDEOPROFILE_H263_BACKWARDCOMPATIBLE	H.263 Version 1 Backward-Compatibility Profile.
XA_VIDEOPROFILE_H263_ISWV2	H.263 Version 2 Interactive and Streaming Wireless Profile.
XA_VIDEOPROFILE_H263_ISWV3	H.263 Version 3 Interactive and Streaming Wireless Profile.
XA_VIDEOPROFILE_H263_HIGHCOMPRESSION	H.263 Conversational High Compression Profile.
XA_VIDEOPROFILE_H263_INTERNET	H.263 Conversational Internet Profile.
XA_VIDEOPROFILE_H263_INTERLACE	H.263 Conversational Interlace Profile.
XA_VIDEOPROFILE_H263_HIGHLATENCY	H.263 High Latency Profile.
XA_VIDEOLEVEL_H263_10	H.263 Level 10.
XA_VIDEOLEVEL_H263_20	H.263 Level 20.
XA_VIDEOLEVEL_H263_30	H.263 Level 30.
XA_VIDEOLEVEL_H263_40	H.263 Level 40.
XA_VIDEOLEVEL_H263_45	H.263 Level 45.
XA_VIDEOLEVEL_H263_50	H.263 Level 50.
XA_VIDEOLEVEL_H263_60	H.263 Level 60.
XA_VIDEOLEVEL_H263_70	H.263 Level 70.

MPEG-4 Profiles and Levels

```
#define XA_VIDEOPROFILE_MPEG4_SIMPLE              ((XAuint32) 0x00000001)
#define XA_VIDEOPROFILE_MPEG4_SIMPLESCALABLE      ((XAuint32) 0x00000002)
#define XA_VIDEOPROFILE_MPEG4_CORE                ((XAuint32) 0x00000003)
#define XA_VIDEOPROFILE_MPEG4_MAIN                ((XAuint32) 0x00000004)
#define XA_VIDEOPROFILE_MPEG4_NBIT                ((XAuint32) 0x00000005)
#define XA_VIDEOPROFILE_MPEG4_SCALABLETEXTURE     ((XAuint32) 0x00000006)
#define XA_VIDEOPROFILE_MPEG4_SIMPLEFACE          ((XAuint32) 0x00000007)
#define XA_VIDEOPROFILE_MPEG4_SIMPLEFBA           ((XAuint32) 0x00000008)
#define XA_VIDEOPROFILE_MPEG4_BASICANIMATED       ((XAuint32) 0x00000009)
#define XA_VIDEOPROFILE_MPEG4_HYBRID              ((XAuint32) 0x0000000A)
#define XA_VIDEOPROFILE_MPEG4_ADVANCEDREALTIME    ((XAuint32) 0x0000000B)
#define XA_VIDEOPROFILE_MPEG4_CORESCALABLE        ((XAuint32) 0x0000000C)
#define XA_VIDEOPROFILE_MPEG4_ADVANCEDCODING      ((XAuint32) 0x0000000D)
#define XA_VIDEOPROFILE_MPEG4_ADVANCEDCORE        ((XAuint32) 0x0000000E)
#define XA_VIDEOPROFILE_MPEG4_ADVANCEDSCALABLE    ((XAuint32) 0x0000000F)

#define XA_VIDEOLEVEL_MPEG4_0    ((XAuint32) 0x00000001)
#define XA_VIDEOLEVEL_MPEG4_0b   ((XAuint32) 0x00000002)
#define XA_VIDEOLEVEL_MPEG4_1    ((XAuint32) 0x00000003)
#define XA_VIDEOLEVEL_MPEG4_2    ((XAuint32) 0x00000004)
#define XA_VIDEOLEVEL_MPEG4_3    ((XAuint32) 0x00000005)
#define XA_VIDEOLEVEL_MPEG4_4    ((XAuint32) 0x00000006)
#define XA_VIDEOLEVEL_MPEG4_4a   ((XAuint32) 0x00000007)
#define XA_VIDEOLEVEL_MPEG4_5    ((XAuint32) 0x00000008)
```

These macros are used for defining MPEG-4 video profiles and levels.

Value	Description
XA_VIDEOPROFILE_MPEG4_SIMPLE	MPEG-4 Simple Profile.
XA_VIDEOPROFILE_MPEG4_SIMPLESCALABLE	MPEG-4 Simple Scalable Profile.
XA_VIDEOPROFILE_MPEG4_CORE	MPEG-4 Core Profile.
XA_VIDEOPROFILE_MPEG4_MAIN	MPEG-4 Main Profile.
XA_VIDEOPROFILE_MPEG4_NBIT	MPEG-4 N-bit Profile.
XA_VIDEOPROFILE_MPEG4_SCALABLETEXTURE	MPEG-4 Scalable Texture Profile.
XA_VIDEOPROFILE_MPEG4_SIMPLEFACE	MPEG-4 Simple Face Animation Profile.
XA_VIDEOPROFILE_MPEG4_SIMPLEFBA	MPEG-4 Simple Face and Body Animation Profile.
XA_VIDEOPROFILE_MPEG4_BASICANIMATED	MPEG-4 Basic Animated Texture Profile.
XA_VIDEOPROFILE_MPEG4_HYBRID	MPEG-4 Hybrid Profile.
XA_VIDEOPROFILE_MPEG4_ADVANCEDREALTIME	MPEG-4 Advanced Real Time Simple Profiles.
XA_VIDEOPROFILE_MPEG4_CORESCALABLE	MPEG-4 Core Scalable Profile.
XA_VIDEOPROFILE_MPEG4_ADVANCEDCODING	MPEG-4 Advanced Coding Efficiency Profile.
XA_VIDEOPROFILE_MPEG4_ADVANCEDCORE	MPEG-4 Advanced Core Profile.
XA_VIDEOPROFILE_MPEG4_ADVANCEDSCALABLE	MPEG-4 Advanced Scalable Texture.

Value	Description
XA_VIDEOLEVEL_MPEG4_0	MPEG-4 Level 0.
XA_VIDEOLEVEL_MPEG4_0b	MPEG-4 Level 0b.
XA_VIDEOLEVEL_MPEG4_1	MPEG-4 Level 1.
XA_VIDEOLEVEL_MPEG4_2	MPEG-4 Level 2.
XA_VIDEOLEVEL_MPEG4_3	MPEG-4 Level 3.
XA_VIDEOLEVEL_MPEG4_4	MPEG-4 Level 4.
XA_VIDEOLEVEL_MPEG4_4a	MPEG-4 Level 4a.
XA_VIDEOLEVEL_MPEG4_5	MPEG-4 Level 5.

AVC Profiles and Levels

```
#define XA_VIDEOPROFILE_AVC_BASELINE   ((XAuint32) 0x00000001)
#define XA_VIDEOPROFILE_AVC_MAIN       ((XAuint32) 0x00000002)
#define XA_VIDEOPROFILE_AVC_EXTENDED   ((XAuint32) 0x00000003)
#define XA_VIDEOPROFILE_AVC_HIGH       ((XAuint32) 0x00000004)
#define XA_VIDEOPROFILE_AVC_HIGH10     ((XAuint32) 0x00000005)
#define XA_VIDEOPROFILE_AVC_HIGH422    ((XAuint32) 0x00000006)
#define XA_VIDEOPROFILE_AVC_HIGH444    ((XAuint32) 0x00000007)

#define XA_VIDEOLEVEL_AVC_1      ((XAuint32) 0x00000001)
#define XA_VIDEOLEVEL_AVC_1B     ((XAuint32) 0x00000002)
#define XA_VIDEOLEVEL_AVC_11     ((XAuint32) 0x00000003)
#define XA_VIDEOLEVEL_AVC_12     ((XAuint32) 0x00000004)
#define XA_VIDEOLEVEL_AVC_13     ((XAuint32) 0x00000005)
#define XA_VIDEOLEVEL_AVC_2      ((XAuint32) 0x00000006)
#define XA_VIDEOLEVEL_AVC_21     ((XAuint32) 0x00000007)
#define XA_VIDEOLEVEL_AVC_22     ((XAuint32) 0x00000008)
#define XA_VIDEOLEVEL_AVC_3      ((XAuint32) 0x00000009)
#define XA_VIDEOLEVEL_AVC_31     ((XAuint32) 0x0000000A)
#define XA_VIDEOLEVEL_AVC_32     ((XAuint32) 0x0000000B)
#define XA_VIDEOLEVEL_AVC_4      ((XAuint32) 0x0000000C)
#define XA_VIDEOLEVEL_AVC_41     ((XAuint32) 0x0000000D)
#define XA_VIDEOLEVEL_AVC_42     ((XAuint32) 0x0000000E)
#define XA_VIDEOLEVEL_AVC_5      ((XAuint32) 0x0000000F)
#define XA_VIDEOLEVEL_AVC_51     ((XAuint32) 0x00000010)
```

These macros are used for defining AVC video profiles and levels.

Value	Description
XA_VIDEOPROFILE_AVC_BASELINE	AVC Baseline Profile.
XA_VIDEOPROFILE_AVC_MAIN	AVC Main Profile.
XA_VIDEOPROFILE_AVC_EXTENDED	AVC Extended Profile.
XA_VIDEOPROFILE_AVC_HIGH	AVC High Profile.
XA_VIDEOPROFILE_AVC_HIGH10	AVC High 10 Profile.
XA_VIDEOPROFILE_AVC_HIGH422	AVC High 4:2:2 Profile.
XA_VIDEOPROFILE_AVC_HIGH444	AVC High 4:4:4 Profile.
XA_VIDEOLEVEL_AVC_1	AVC Level 1.
XA_VIDEOLEVEL_AVC_1B	AVC Level 1b.
XA_VIDEOLEVEL_AVC_11	AVC Level 1.1.
XA_VIDEOLEVEL_AVC_12	AVC Level 1.2.
XA_VIDEOLEVEL_AVC_13	AVC Level 1.3.
XA_VIDEOLEVEL_AVC_2	AVC Level 2.
XA_VIDEOLEVEL_AVC_21	AVC Level 2.1.
XA_VIDEOLEVEL_AVC_22	AVC Level 2.2.
XA_VIDEOLEVEL_AVC_3	AVC Level 3.
XA_VIDEOLEVEL_AVC_31	AVC Level 3.1.
XA_VIDEOLEVEL_AVC_32	AVC Level 3.2.
XA_VIDEOLEVEL_AVC_4	AVC Level 4.
XA_VIDEOLEVEL_AVC_41	AVC Level 4.1.
XA_VIDEOLEVEL_AVC_42	AVC Level 4.2.
XA_VIDEOLEVEL_AVC_5	AVC Level 5.
XA_VIDEOLEVEL_AVC_51	AVC Level 5.1.

VC-1 Profiles and Levels

```
#define XA_VIDEOPROFILE_VC1_SIMPLE     ((XAuint32) 0x00000001)
#define XA_VIDEOPROFILE_VC1_MAIN       ((XAuint32) 0x00000002)
#define XA_VIDEOPROFILE_VC1_ADVANCED   ((XAuint32) 0x00000003)

#define XA_VIDEOLEVEL_VC1_LOW    ((XAuint32) 0x00000001)
#define XA_VIDEOLEVEL_VC1_MEDIUM ((XAuint32) 0x00000002)
#define XA_VIDEOLEVEL_VC1_HIGH   ((XAuint32) 0x00000003)
#define XA_VIDEOLEVEL_VC1_L0     ((XAuint32) 0x00000004)
#define XA_VIDEOLEVEL_VC1_L1     ((XAuint32) 0x00000005)
#define XA_VIDEOLEVEL_VC1_L2     ((XAuint32) 0x00000006)
#define XA_VIDEOLEVEL_VC1_L3     ((XAuint32) 0x00000007)
#define XA_VIDEOLEVEL_VC1_L4     ((XAuint32) 0x00000008)
```

These macros are used for defining VC-1 video profiles and levels.

Value	Description
XA_VIDEOPROFILE_VC1_SIMPLE	VC-1 Simple Profile.
XA_VIDEOPROFILE_VC1_MAIN	VC-1 Main Profile.
XA_VIDEOPROFILE_VC1_ADVANCED	VC-1 Advanced Profile.
XA_VIDEOLEVEL_VC1_LOW	VC-1 Level Low.
XA_VIDEOLEVEL_VC1_MEDIUM	VC-1 Level Medium.
XA_VIDEOLEVEL_VC1_HIGH	VC-1 Level High.
	VC-1 Level L0.
XA_VIDEOLEVEL_VC1_L1	VC-1 Level L1.
XA_VIDEOLEVEL_VC1_L2	VC-1 Level L2.
XA_VIDEOLEVEL_VC1_L3	VC-1 Level L3.
XA_VIDEOLEVEL_VC1_L4	VC-1 Level L4.

VP8 Profiles and Levels

```
#define XA_VIDEOPROFILE_VP8_MAIN       ((XAuint32) 0x00000001)

#define XA_VIDEOLEVEL_VP8_VERSION0     ((XAuint32) 0x00000001)
#define XA_VIDEOLEVEL_VP8_VERSION1     ((XAuint32) 0x00000002)
#define XA_VIDEOLEVEL_VP8_VERSION2     ((XAuint32) 0x00000003)
#define XA_VIDEOLEVEL_VP8_VERSION3     ((XAuint32) 0x00000004)
```

These macros are used for defining VP8 video profiles and levels.

Value	Description
XA_VIDEOPROFILE_VP8_MAIN	VP8 Main Profile.
XA_VIDEOLEVEL_VP8_VERSION0	VP8 Level “Version 0”

Value	Description
XA_VIDEOLEVEL_VP8_VERSION1	VP8 Level “Version 1”.
XA_VIDEOLEVEL_VP8_VERSION2	VP8 Level “Version 2”.
XA_VIDEOLEVEL_VP8_VERSION3	VP8 Level “Version 3”.

9.2.80 XA_VIDEOSCALE

```
#define XA_VIDEOSCALE_STRETCH    ((XAuint32) 0x00000001)
#define XA_VIDEOSCALE_FIT        ((XAuint32) 0x00000002)
#define XA_VIDEOSCALE_CROP       ((XAuint32) 0x00000003)
```

These macros are used to select the video scaling option. They are intended to be used with XAVideoPostProcessingItf.

Value	Description
XA_VIDEOSCALE_STRETCH	The source and destination rectangle’s width and height parameters are used to calculate the scaling factors independently. Aspect ratio is ignored.
XA_VIDEOSCALE_FIT	The minimum scale factor between the destination rectangle’s width over the source rectangle’s width and the destination rectangle’s height over the source rectangle’s height is used. Aspect ratio is maintained. Frame is centered.
XA_VIDEOSCALE_CROP	The maximum scale factor between the destination rectangle’s width over the source rectangle’s width and the destination rectangle’s height over the source rectangle’s height is used. Aspect ratio is maintained. Frame is centered.

Part 3: Appendices

Appendix A: References

CP	OpenMAX Content Pipe Application Programming Interface Specification, The Khronos Group
DLS2	Downloadable Sounds Level 2.1 Specification (RP-025/Amd1), MIDI Manufacturers Association, Los Angeles, CA, USA, January 2001.
ISO639	Language codes, http://www.iso.org/iso/en/prods-services/popstds/languagecodes.html, ISO 639.
ISO1000	SI units and recommendations for the use of their multiples and of certain other units, ISO 1000:1992, 2003.
ISO3166	Country name codes, http://www.iso.org/iso/en/prods-services/popstds/countrynamecodes.html, ISO 3166-1:2006
JSR135	JSR-135: Mobile Media API (http://www.jcp.org/en/jsr/detail?id=135).
mDLS	Mobile DLS Specification, RP-041, MIDI Manufacturers Association, Los Angeles, CA, USA, 2003.
MIDI	The Complete MIDI 1.0 Detailed Specification, Document version 96.1, MIDI Manufacturers Association, Los Angeles, CA, USA, 1996 (Contains MIDI 1.0 Detailed Specification, MIDI Time Code, Standard MIDI Files 1.0, General MIDI System Level 1, MIDI Show Control 1.1, and MIDI Machine Control)
MPEG1	ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 11172-3 "Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s, Part 3: Audio", 1993
MPEG2	ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 13818-3 "Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part 3: Audio", 1998.
mXMF	Mobile XMF Content Format Specification, RP-042. MIDI Manufacturers Association, Los Angeles, CA, USA, September 2004.
RFC3066	Tags for the Identifications of Languages, http://tools.ietf.org/html/rfc3066, RFC-3066, IETF, 2001.
SP-MIDI	Scalable Polyphony MIDI Specification (RP-034), MIDI Manufacturers Association, Los Angeles, CA, USA, December 2001.
UUID	Information Technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and Registration of Universally Unique Identifiers (UUIDs) and their Use as ASN.1 Object Identifier Components, ITU-T Rec. X.667 | ISO/IEC 9834-8, 2004
OSLES	Open Sound Library for Embedded Systems (OpenSL ES) Application Programming Interface Specification, Khronos Group.

Appendix B: Glossary of RDS Terms

This glossary is informative, not normative. Please see the RDS specification for accurate definitions of these terms.

Acronym	Term	Description
AF	Alternate Frequency.	Information about other frequencies broadcasting the same programme.
CT	Clock Time and date.	Current time and date
EON	Enhanced Other Networks information.	Information about other programme services and their content.
ODA	Open Data Application.	Usages of RDS technology that are not explicitly specified by the RDS standard.
PI	Programme Identification code	The PI is not meant to directly display to the end user, but to uniquely identify a programme. This can be used for operations such as detecting that two frequencies are transmitting the same programme.
PTY	Programme TYpe code	A code that specifies the current programme type.
RT	Radio Text	Text transmission.
RT+	Radio Text Plus	RT+ gives access to specific elements (RT+ content type classes) of Radio Text messages. Examples are song title, news, studio telephone number and broadcaster's web address.
TA	Traffic Announcement code	A Boolean indicating when a traffic announcement is currently on air. See TP.
TP	Traffic Programme identification code	A flag telling that the programme carries traffic announcements. See TA.

Appendix C: Object-Interface Mapping

The following table describes the object-interface mapping per profile. It also shows mandated objects for each profile in its second row.

Object	Engine		Media Player		Media Recorder		Radio		Camera		Output Mix		Vibra		LED Array		Metadata Extractor	
Interface	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**
XAAudioDecoderCapabilitiesItf																		
XAAudioEncoderCapabilitiesItf						2												
XAAudioEncoderItf						2												
XAAudioIODeviceCapabilitiesItf																		
XACameraItf																		
XACameraCapabilitiesItf																		
XAConfigExtensionsItf																		
XADeviceVolumeItf																		
XADynamicInterfaceManagementItf																		
XADynamicSourceItf																		
XADynamicSourceSinkChangeItf																		
XAEngineItf																		
XAEqualizerItf																		
XAImageControlsItf																		
XAImageDecoderCapabilitiesItf																		
XAImageEffectsItf																		
XAImageEncoderCapabilitiesItf																		
XAImageEncoderItf						3												
XALEDArrayItf																		

Object	Engine		Media Player		Media Recorder		Radio		Camera		Output Mix		Vibra		LED Array		Metadata Extractor	
Interface	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**	**MP**	**MR**
XAMetadataExtractionItf																		
XAMetadataInsertionItf																		
XAMetadataMessageItf																		
XAMetadataTraversalItf																		
XAObjectItf																		
XAOutputMixItf																		
XAPlayItf																		
XAPlaybackRateItf			1	1														
XAPrefetchStatusItf																		
XARadioItf																		
XARDSItf																		
XARecordItf						2												
XASeekItf			2	2														
XASnapShotItf						3												
XAStreamInformationItf																		
XAThreadSyncItf																		
XAVibraItf																		
XAVideoDecoderCapabilitiesItf																		
XAVideoEncoderCapabilitiesItf																		
XAVideoEncoderItf						2												
XAVideoPostProcessingItf																		
XAVolumeItf			4	4														

Legend	
MP	Object mandated in Media Player profile
MR	Object mandated in Media Player/Recorder profile
MP	Object optional in Media Player profile
MR	Object optional in Media Player/Recorder profile
	Implicit and mandated interface
	Mandated (explicit) interface
1	Mandated (explicit) interface with some optional methods, see comments.
	Applicable optional interfaces

Comments for mandated interfaces:

1. Mandated only for timed-based media content stored locally.
2. Mandated only for uses cases with audio or video.
3. Mandated only for uses cases with image.
4. Mandated only for uses cases with audio.

Appendix D: Sample Code

This appendix provides sample code illustrating how objects can be used together to support simple use cases. The sample code shows how to use the API and is for purposes of illustration only – it is not intended to provide realistic application code and these code fragments are not necessarily complete.

D.1 Audio Playback with Equalizer

```
/*
 *  OpenMAX AL - Audio Playback with Equalizer
 */

#include <stdio.h>
#include <stdlib.h>

#include <OMXAL/OpenMAXAL.h>

#define MAX_NUMBER_INTERFACES 5

/* Global variables. (Should be local in real application.) */
XAObjectItf    engine; /* OpenMAX AL Engine */
XAObjectItf    player;
XAObjectItf    outputMix;
XAPlayItf      playItf;
XAEqualizerItf equalizerItf;

/* Checks for error. If any errors exit the application! */
void CheckErr (XAresult res)
{
    if (res != XA_RESULT_SUCCESS)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
}

/*
 * Draws single EQ band to the screen. Called by drawEQDisplay
 */
void drawEQBand (int minFreq, int maxFreq, int level)
{
    /* Insert drawing routines here for single EQ band.  (Use
     * GetBandLevelRange and screen height to map the level to screen
     * y-coordinate.) */
}

/*
 * Called when the display is repainted.
 */
void drawEQDisplay (void)
{
    XAuint16      numBands;
    XAmillibel   bandLevel;
    XAmillibel   minLevel;
```

```
    XAmillibel   maxLevel;
    XAmilliHertz minFreq;
    XAmilliHertz maxFreq;
    int          band;
    XAresult     res;

    res = (*equalizerItf)->GetNumberOfBands(equalizerItf,
            &numBands); CheckErr(res);
    res = (*equalizerItf)->GetBandLevelRange(equalizerItf,
            &minLevel, &maxLevel); CheckErr(res);

    for (band = 0; band<numBands; band++)
    {
        res = (*equalizerItf)->GetBandFreqRange(equalizerItf,
                (XAint16) band, &minFreq, &maxFreq); CheckErr(res);
        res = (*equalizerItf)->GetBandLevel(equalizerItf,
                (XAint16)band, &bandLevel); CheckErr(res);
        drawEQBand(minFreq, maxFreq, bandLevel);
    }
}

/*
 * Initializes the OpenMAX AL engine, starts the playback of some
 * music from a file and draws the graphical equalizer
 */
void init (void)
{
    XAEngineItf             EngineItf;
    XADataSource            audioSource;
    XADataLocator_URI       uri;
    XADataFormat_MIME       mime;
    XADataSink              audioSink;
    XADataLocator_OutputMix locator_outputmix;
    XAVolumeItf             volumeItf;
    XAresult                res;

    int i;

    XAboolean     required[MAX_NUMBER_INTERFACES];
    XAInterfaceID iidArray[MAX_NUMBER_INTERFACES];


   XAEngineOption EngineOption[] = {
         (XAuint32) XA_ENGINEOPTION_THREADSAFE,
         (XAuint32) XA_BOOLEAN_TRUE,
         (XAuint32) XA_ENGINEOPTION_MAJORVERSION, (XAuint32) 1,
         (XAuint32) XA_ENGINEOPTION_MINORVERSION, (XAuint32) 1
   };


    /* Create OpenMAX AL */
    res = xaCreateEngine(&engine,
            3, EngineOption, 0, NULL, NULL); CheckErr(res);

    /* Realizing the XA Engine in synchronous mode. */
    res = (*engine)->Realize(engine,
```

```
        XA_BOOLEAN_FALSE); CheckErr(res);

/* Get the XA Engine Interface which is implicit */
res = (*engine)->GetInterface(engine,
        XA_IID_ENGINE, (void*) &EngineItf); CheckErr(res);

/* Initialize arrays required[] and iidArray[] */
for (i = 0; i < MAX_NUMBER_INTERFACES; i++)
{
    required[i] = XA_BOOLEAN_FALSE;
    iidArray[i] = XA_IID_NULL;
}

/* Set arrays required[] and iidArray[] for VOLUME and EQUALIZER
 * interfaces */
required[0] = XA_BOOLEAN_TRUE;
iidArray[0] = XA_IID_VOLUME;
required[1] = XA_BOOLEAN_TRUE;
iidArray[1] = XA_IID_EQUALIZER;

/* Create Output Mix object to be used by player */
res = (*EngineItf)->CreateOutputMix(EngineItf,
        &outputMix, 2, iidArray, required); CheckErr(res);

/* Realizing the Output Mix object in synchronous mode. */
res = (*outputMix)->Realize(outputMix,
        XA_BOOLEAN_FALSE); CheckErr(res);

/* Get play and equalizer interfaces */
res = (*outputMix)->GetInterface(outputMix,
        XA_IID_VOLUME, (void*) &volumeItf); CheckErr(res);
res = (*outputMix)->GetInterface(outputMix,
        XA_IID_EQUALIZER, (void*) &equalizerItf); CheckErr(res);

/* Setup the data source structure */
uri.locatorType      = XA_DATALOCATOR_URI;
uri.pURI              = (XAchar *) "file:///music.wav";
mime.formatType      = XA_DATAFORMAT_MIME;
mime.pMimeType        = (XAchar *) "audio/x-wav";
mime.containerType   = XA_CONTAINERTYPE_WAV;
audioSource.pLocator = (void*) &uri;
audioSource.pFormat  = (void*) &mime;

/* Setup the data sink structure */
locator_outputmix.locatorType = XA_DATALOCATOR_OUTPUTMIX;
locator_outputmix.outputMix   = outputMix;
audioSink.pLocator            = (void*) &locator_outputmix;
audioSink.pFormat             = NULL;

/* Set arrays required[] and iidArray[] for no interfaces (PlayItf
 * is implicit) */
required[0] = XA_BOOLEAN_FALSE;
iidArray[0] = XA_IID_NULL;
required[1] = XA_BOOLEAN_FALSE;
iidArray[1] = XA_IID_NULL;
```

```
    /* Create the music player */
    res = (*EngineItf)->CreateMediaPlayer(EngineItf,
            &player, &audioSource, NULL, &audioSink, NULL, NULL, NULL,
            0, iidArray, required); CheckErr(res);

    /* Realizing the player in synchronous mode. */
    res = (*player)->Realize(player, XA_BOOLEAN_FALSE); CheckErr(res);

    /* Get the play interface */
    res = (*player)->GetInterface(player,
            XA_IID_PLAY, (void*) &playItf); CheckErr(res);

    /* Before we start set volume to -3dB (-300mB) and enable equalizer */
    res = (*volumeItf)->SetVolumeLevel(volumeItf, -300); CheckErr(res);
    res = (*equalizerItf)->SetEnabled(equalizerItf,
            XA_BOOLEAN_TRUE); CheckErr(res);

    /* Play the music */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_PLAYING); CheckErr(res);

    /* Draw the graphical EQ */
    drawEQDisplay();
}

/*
 * Shuts down the OpenMAX AL engine.
 */
void destroy (void)
{
    XAresult res;

    /* Stop the music */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_STOPPED); CheckErr(res);

    /* Destroy the player */
    (*player)->Destroy(player);

    /* Destroy Output Mix object */
    (*outputMix)->Destroy(outputMix);

    /* Shutdown OpenMAX AL */
    (*engine)->Destroy(engine);
}

/*
 * Called by UI when user increases or decreases a band level.
 */
void setBandLevel(XAint16 band, XAboolean increase)
{
    XAuint16    numBands;
    XAmillibel bandLevel;
    XAmillibel minLevel;
    XAmillibel maxLevel;
```

```
    XAresult res;

    res = (*equalizerItf)->GetNumberOfBands(equalizerItf,
            &numBands); CheckErr(res);
    res = (*equalizerItf)->GetBandLevelRange(equalizerItf,
            &minLevel, &maxLevel); CheckErr(res);

    if (band >= numBands)
    {
        /* Error. Insert debug print here. */
        exit(0);
    }

    res = (*equalizerItf)->GetBandLevel(equalizerItf,
            band, &bandLevel); CheckErr(res);

    if (increase==XA_BOOLEAN_TRUE)
    {
        /* increase the level by 1 dB (100mB) if the max supported level
         * is not exceeded */
        bandLevel = bandLevel + 100;
        if(bandLevel < maxLevel)
        {
            res = (*equalizerItf)->SetBandLevel(equalizerItf,
                    band, bandLevel); CheckErr(res);
            drawEQDisplay();
        }
    }
    else /* increase==false */
    {
        /* decrease the level by 1 dB (100mB) if the min supported level
         * is not crossed */
        bandLevel = bandLevel - 100;
        if( bandLevel > minLevel )
        {
            res = (*equalizerItf)->SetBandLevel(equalizerItf,
                    band, bandLevel); CheckErr(res);
            drawEQDisplay();
        }
    }
}
```

D.2 Audio/Video Playback

```
/*
 *  OpenMAX AL - Audio/Video Playback Example
 */

#include <stdio.h>
#include <stdlib.h>

#include <OMXAL/OpenMAXAL.h>

#define MAX_NUMBER_INTERFACES     5
#define MAX_NUMBER_OUTPUT_DEVICES 3
```

```
#define POSITION_UPDATE_PERIOD    1000 /* 1 sec */



/* Checks for error. If any errors exit the application! */
void CheckErr (XAresult res)
{
    if (res != XA_RESULT_SUCCESS)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
}

void PlayEventCallback (
        XAPlayItf caller,
        void *    pContext,
        XAuint32  playevent)
{
    /* Callback code goes here */
}

/*
 * Test audio/video playback from a 3GPP file.
 *
 * NOTE: For the purposes of this example, the implementation is assumed
 * to support the requisite audio and video codecs. Therefore, video and
 * audio decoder capabilities are NOT checked in this example.
 */
void TestAudioVideoPlayback (XAObjectItf engine)
{
    XAObjectItf                   player;
    XAObjectItf                   OutputMix;
    XAPlayItf                     playItf;
    XAEngineItf                   EngineItf;
    XAAudioIODeviceCapabilitiesItf AudioIODeviceCapabilitiesItf;
    XAAudioOutputDescriptor       AudioOutputDescriptor;
    XAresult                      res;

    XADataSink                  audioSink;
    XADataSink                  videoSink;
    XADataLocator_OutputMix     locator_outputmix;
    XADataLocator_NativeDisplay locator_displayregion;
    XAVolumeItf                 volumeItf;

    XADataSource      avSource;
    XADataLocator_URI uri;
    XADataFormat_MIME mime;

    int  i;
    char c;

    XAboolean     required[MAX_NUMBER_INTERFACES];
    XAInterfaceID iidArray[MAX_NUMBER_INTERFACES];

    XAuint32       OutputDeviceIDs[MAX_NUMBER_OUTPUT_DEVICES];
```

```
    XAint32        numOutputs          = 0;
    XAboolean      hfs_available       = XA_BOOLEAN_FALSE;
    XAboolean      hfs_default         = XA_BOOLEAN_FALSE;
    XAuint32       hfs_deviceID        = 0;
    XANativeHandle nativeWindowHandle  = NULL;
    XANativeHandle nativeDisplayHandle = NULL;

    /* Get the XA Engine Interface, which is implicit */
    res = (*engine)->GetInterface(engine,
            XA_IID_ENGINE, (void*) &EngineItf); CheckErr(res);

    /* Get the Audio IO DEVICE CAPABILITIES interface, which is also
     * implicit */
    res = (*engine)->GetInterface(engine,
            XA_IID_AUDIOIODEVICECAPABILITIES,
            (void*) &AudioIODeviceCapabilitiesItf); CheckErr(res);

    numOutputs = MAX_NUMBER_OUTPUT_DEVICES;

    res = (*AudioIODeviceCapabilitiesItf)->
        GetAvailableAudioOutputs(AudioIODeviceCapabilitiesItf,
                &numOutputs, OutputDeviceIDs); CheckErr(res);

    /* Search for integrated handsfree loudspeaker */
    for (i = 0; i < numOutputs; i++)
    {
        res = (*AudioIODeviceCapabilitiesItf)->
            QueryAudioOutputCapabilities(AudioIODeviceCapabilitiesItf,
                    OutputDeviceIDs[i], &AudioOutputDescriptor);
        CheckErr(res);
        if ((AudioOutputDescriptor.deviceConnection ==
                    XA_DEVCONNECTION_INTEGRATED) &&
                (AudioOutputDescriptor.deviceScope ==
                 XA_DEVSCOPE_ENVIRONMENT) &&
                (AudioOutputDescriptor.deviceLocation ==
                 XA_DEVLOCATION_HANDSET))
        {
            hfs_deviceID  = OutputDeviceIDs[i];
            hfs_available = XA_BOOLEAN_TRUE;
            break;
        }
    }

    /* If preferred output audio device is not available, no point in
     * continuing */
    if (!hfs_available)
    {
        /* Appropriate error message here */
        exit(1);
    }

    numOutputs = MAX_NUMBER_OUTPUT_DEVICES;
    res = (*AudioIODeviceCapabilitiesItf)->
        GetDefaultAudioDevices(AudioIODeviceCapabilitiesItf,
                XA_DEFAULTDEVICEID_AUDIOOUTPUT, &numOutputs,
OutputDeviceIDs);
```

```
    CheckErr(res);

    /* Check whether Default Output devices include the handsfree
     * loudspeaker */
    for (i = 0; i < numOutputs; i++)
    {
        if (OutputDeviceIDs[i] == hfs_deviceID)
        {
            hfs_default = XA_BOOLEAN_TRUE;
            break;
        }
    }

    /* Expect handsfree loudspeaker to be set as one of the default
     * output devices */
    if (!hfs_default)
    {
        /* Debug printing to be placed here */
        exit(1);
    }

    /* Initialize arrays required[] and iidArray[] */
    for (i = 0; i < MAX_NUMBER_INTERFACES; i++)
    {
        required[i] = XA_BOOLEAN_FALSE;
        iidArray[i] = XA_IID_NULL;
    }

    /* Set arrays required[] and iidArray[] for VOLUME interface */
    required[0] = XA_BOOLEAN_TRUE;
    iidArray[0] = XA_IID_VOLUME;

    /* Create Output Mix object to be used by player */
    res = (*EngineItf)->CreateOutputMix(EngineItf,
            &OutputMix, 1, iidArray, required); CheckErr(res);

    /* Realizing the Output Mix object in synchronous mode  */
    res = (*OutputMix)->Realize(OutputMix,
            XA_BOOLEAN_FALSE); CheckErr(res);

    /* Get the volume interface on the output mix */
    res = (*OutputMix)->GetInterface(OutputMix,
            XA_IID_VOLUME, (void*)&volumeItf); CheckErr(res);

    /* Setup the audio/video data source structure */
    uri.locatorType    = XA_DATALOCATOR_URI;
    uri.pURI            = (XAchar *) "file:///avmedia.3gp";
    mime.formatType    = XA_DATAFORMAT_MIME;
    mime.pMimeType      = (XAchar *) "video/3gpp";
    mime.containerType = XA_CONTAINERTYPE_3GPP; /* provided as a hint to
                                                 * the player */
    avSource.pLocator = (void*) &uri;
    avSource.pFormat  = (void*) &mime;

    /* Setup the audio data sink structure */
    locator_outputmix.locatorType = XA_DATALOCATOR_OUTPUTMIX;
```

```
    locator_outputmix.outputMix   = OutputMix;
    audioSink.pLocator            = (void*) &locator_outputmix;
    audioSink.pFormat             = NULL;

    /* Set nativeWindowHandle and nativeDisplayHandle to
     * platform-specific values here */
    /* nativeWindowHandle  = <a platform-specific value>; */
    /* nativeDisplayHandle = <a platform-specific value>; */

    /* Setup the video data sink structure */
    locator_displayregion.locatorType = XA_DATALOCATOR_NATIVEDISPLAY;
    locator_displayregion.hWindow     = nativeWindowHandle;
    locator_displayregion.hDisplay    = nativeDisplayHandle;
    videoSink.pLocator                = (void*) &locator_displayregion;
    videoSink.pFormat                 = NULL;

    /* Create the media player. pBankSrc is NULL as we have a non-MIDI
     * data source */
    res = (*EngineItf)->CreateMediaPlayer(EngineItf,
            &player, &avSource, NULL, &audioSink, &videoSink, NULL, NULL,
            1, iidArray, required); CheckErr(res);

    /* Realizing the player in synchronous mode */
    res = (*player)->Realize(player, XA_BOOLEAN_FALSE); CheckErr(res);

    /* Get play interface */
    res = (*player)->GetInterface(player,
            XA_IID_PLAY, (void*) &playItf); CheckErr(res);

    /* Setup to receive position event callbacks */
    res = (*playItf)->RegisterCallback(playItf,
            PlayEventCallback, NULL); CheckErr(res);

    /* Set notifications to occur after every 1 second - might be useful
     * in updating a progress bar */
    res = (*playItf)->SetPositionUpdatePeriod(playItf,
            POSITION_UPDATE_PERIOD); CheckErr(res);
    res = (*playItf)->SetCallbackEventsMask(playItf,
            XA_PLAYEVENT_HEADATNEWPOS); CheckErr(res);

    /* Before we start, set volume to -3dB (-300mB) */
    res = (*volumeItf)->SetVolumeLevel(volumeItf, -300); CheckErr(res);

    /* Play the media */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_PLAYING); CheckErr(res);

    while ((c = getchar()) != 'q')
    {
        XAuint32 playState;

        switch(c)
        {
            case '1':
                /* Play the media - if it is not already playing */
                res = (*playItf)->GetPlayState(playItf,
```

```
                        &playState); CheckErr(res);
                if (playState != XA_PLAYSTATE_PLAYING)
                {
                    res = (*playItf)->SetPlayState(playItf,
                            XA_PLAYSTATE_PLAYING); CheckErr(res);
                }
                break;

            case '2':
                /* Pause the media - if it is playing */
                res = (*playItf)->GetPlayState(playItf,
                        &playState); CheckErr(res);
                if (playState == XA_PLAYSTATE_PLAYING)
                {
                    res = (*playItf)->SetPlayState(playItf,
                            XA_PLAYSTATE_PAUSED); CheckErr(res);
                }
                break;

            default:
                break;
        }
    }

    /* Stop the media playback */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_STOPPED); CheckErr(res);

    /* Destroy the player object */
    (*player)->Destroy(player);

    /* Destroy the output mix object */
    (*OutputMix)->Destroy(OutputMix);
}

int xa_main (void)
{
    XAresult    res;
    XAObjectItf engine;

    /* Create OpenMAX AL engine in thread-safe mode */

   XAEngineOption EngineOption[] = {
         (XAuint32) XA_ENGINEOPTION_THREADSAFE,
         (XAuint32) XA_BOOLEAN_TRUE,
         (XAuint32) XA_ENGINEOPTION_MAJORVERSION, (XAuint32) 1,
         (XAuint32) XA_ENGINEOPTION_MINORVERSION, (XAuint32) 1
   };

    res = xaCreateEngine(&engine,
            3, EngineOption, 0, NULL, NULL); CheckErr(res);

    /* Realizing the AL Engine in synchronous mode */
    res = (*engine)->Realize(engine, XA_BOOLEAN_FALSE); CheckErr(res);

    TestAudioVideoPlayback(engine);
```

```
    /* Shutdown OpenMAX AL engine */
    (*engine)->Destroy(engine);

    exit(0);
}
```

D.3 Radio with RDS Support

```
/*
 *  OpenMAX AL - Radio with RDS Support
 *
 *  This simple example turns on the radio playback and lets the user to
 *  tune the frequency. It displays the Programme Service name on the
 *  screen.  Assumes that the device supports Radio with RadioItf and
 *  RDSItf.
 */

#include <stdio.h>
#include <stdlib.h>

#include <OMXAL/OpenMAXAL.h>

#define MAX_NUMBER_INTERFACES 3

/* Global variables. (Should be local in real application.) */
XAObjectItf engine; /*OpenMAX AL Engine */
XAObjectItf player;
XAObjectItf outputMix;
XAObjectItf radio;
XAPlayItf   playItf;
XARadioItf  radioItf;
XARDSItf    rdsItf;

XAuint32 currentFreq;
XAuint32 minFreq;
XAuint32 maxFreq;
XAuint32 freqInterval;

/* Dummy semaphore and event related types, prototypes and defines */

typedef XAuint16 Sem_t;  /* System semaphore type would replace Sem_t */

Sem_t semFreq;
Sem_t semFreqRange;

void sem_post (Sem_t * pSemaphore)
{
    /* Implementation specific semaphore post */
}

void sem_wait (Sem_t * pSemaphore)
{
    /* Implementation specific semaphore wait */
}
```

```
/* Checks for error. If any errors exit the application! */
void CheckErr (XAresult res)
{
    if (res != XA_RESULT_SUCCESS)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
}

void RadioCallback(
        XARadioItf caller,
        void *     pContext,
        XAuint32   event,
        XAuint32   eventIntData,
        XAboolean  eventBooleanData)
{
    if (event == XA_RADIO_EVENT_FREQUENCY_CHANGED)
    {
        sem_post(&semFreq);
    }
    else if (event == XA_RADIO_EVENT_FREQUENCY_RANGE_CHANGED)
    {
        sem_post(&semFreqRange);
    }
}

void RDSCallback(
        XARDSItf caller,
        void *   pContext,
        XAuint16 event,
        XAuint8  eventData)
{
    if (event == XA_RDS_EVENT_NEW_PS)
    {
        /* update Programme Service name on the screen by querying it
         * from GetProgrammeServiceName */
    }
}

/*
 * Initializes the OpenMAX AL engine and starts the playback of radio
 */
void init (void)
{
    XAEngineItf             EngineItf;
    XADataSource            audioSource;
    XADataLocator_IODevice  locatorIODevice;
    XADataSink              audioSink;
    XADataLocator_OutputMix locator_outputmix;
    XAVolumeItf             volumeItf;
    XAresult                res;

    int       i;
    XAboolean supported;
```

```
 XAboolean     required[MAX_NUMBER_INTERFACES];
 XAInterfaceID iidArray[MAX_NUMBER_INTERFACES];


XAEngineOption EngineOption[] = {
      (XAuint32) XA_ENGINEOPTION_THREADSAFE,
      (XAuint32) XA_BOOLEAN_TRUE,
      (XAuint32) XA_ENGINEOPTION_MAJORVERSION, (XAuint32) 1,
      (XAuint32) XA_ENGINEOPTION_MINORVERSION, (XAuint32) 1
};


 /* CREATE ENGINE */

 /* Create OpenMAX AL */
 res = xaCreateEngine(&engine,
         3, EngineOption, 0, NULL, NULL); CheckErr(res);

 /* Realizing the XA Engine in synchronous mode. */
 res = (*engine)->Realize(engine, XA_BOOLEAN_FALSE); CheckErr(res);

 /* Get the XA Engine Interface which is implicit */
 res = (*engine)->GetInterface(engine,
         XA_IID_ENGINE, (void*) &EngineItf); CheckErr(res);

 /* Initialize arrays required[] and iidArray[] */
 for (i = 0; i < MAX_NUMBER_INTERFACES; i++)
 {
     required[i] = XA_BOOLEAN_FALSE;
     iidArray[i] = XA_IID_NULL;
 }

 /* CREATE OUTPUTMIX */

 /* Set arrays required[] and iidArray[] for VOLUME interface */
 required[0] = XA_BOOLEAN_TRUE;
 iidArray[0] = XA_IID_VOLUME;

 /* Create Output Mix object to be used by player */
 res = (*EngineItf)->CreateOutputMix(EngineItf,
         &outputMix, 1, iidArray, required); CheckErr(res);

 /* Realizing the Output Mix object in synchronous mode. */
 res = (*outputMix)->Realize(outputMix,
         XA_BOOLEAN_FALSE); CheckErr(res);

 /* Get play and equalizer interfaces */
 res = (*outputMix)->GetInterface(outputMix,
         XA_IID_VOLUME, (void*) &volumeItf); CheckErr(res);

 /* CREATE RADIO */

 /* Set arrays required[] and iidArray[] for RDS interface (RadioItf
  * is implicit) */
 required[0] = XA_BOOLEAN_TRUE;
```

```
    iidArray[0] = XA_IID_RDS;

    /* Create Radio object to be used by player */
    res = (*EngineItf)->CreateRadioDevice(EngineItf,
            &radio, 1, iidArray, required); CheckErr(res);

    /* Realizing the Radio object in synchronous mode. */
    res = (*outputMix)->Realize(radio, XA_BOOLEAN_FALSE); CheckErr(res);

    /* Get play and equalizer interfaces */
    res = (*radio)->GetInterface(radio,
            XA_IID_RADIO, (void*) &radioItf); CheckErr(res);
    res = (*radio)->GetInterface(radio,
            XA_IID_RDS, (void*) &rdsItf); CheckErr(res);

    /* Register callbacks */
    res = (*radioItf)->RegisterRadioCallback(radioItf,
            RadioCallback, NULL); CheckErr(res);
    res = (*rdsItf)->RegisterRDSCallback(rdsItf,
            RDSCallback, NULL); CheckErr(res);

    /* Setup the data source structure */
    locatorIODevice.locatorType = XA_DATALOCATOR_IODEVICE;
    locatorIODevice.deviceType  = XA_IODEVICE_RADIO;
    locatorIODevice.deviceID    = 0;  /* ignored */
    locatorIODevice.device      = radio;
    audioSource.pLocator        = (void*) &locatorIODevice;
    audioSource.pFormat         = NULL;

    /* Setup the data sink structure */
    locator_outputmix.locatorType = XA_DATALOCATOR_OUTPUTMIX;
    locator_outputmix.outputMix   = outputMix;
    audioSink.pLocator            = (void*) &locator_outputmix;
    audioSink.pFormat             = NULL;

    /* CREATE PLAYER */

    /* Set arrays required[] and iidArray[] for no interfaces (PlayItf
     * is implicit) */
    required[0] = XA_BOOLEAN_FALSE;
    iidArray[0] = XA_IID_NULL;

    /* Create the music player */
    res = (*EngineItf)->CreateMediaPlayer(EngineItf,
            &player, &audioSource, NULL, &audioSink, NULL, NULL, NULL,
            0, iidArray, required); CheckErr(res);

    /* Realizing the player in synchronous mode. */
    res = (*player)->Realize(player, XA_BOOLEAN_FALSE); CheckErr(res);

    /* Get the play interface */
    res = (*player)->GetInterface(player,
            XA_IID_PLAY, (void*) &playItf); CheckErr(res);

    /* SETUP */
```

```
    /* Before we start set volume to -3dB (-300mB) */
    res = (*volumeItf)->SetVolumeLevel(volumeItf, -300); CheckErr(res);

    /* Set up the radio frequency range */
    res = (*radioItf)->IsFreqRangeSupported(radioItf,
            XA_FREQRANGE_FMEUROAMERICA, &supported); CheckErr(res);
    if (supported)
    {
        res = (*radioItf)->GetFreqRangeProperties(radioItf,
                XA_FREQRANGE_FMEUROAMERICA, &minFreq, &maxFreq,
                &freqInterval); CheckErr(res);
        res = (*radioItf)->SetFreqRange(radioItf,
                XA_FREQRANGE_FMEUROAMERICA); CheckErr(res);
    }
    else
    {
        res = (*radioItf)->IsFreqRangeSupported(radioItf,
                XA_FREQRANGE_FMJAPAN, &supported); CheckErr(res);
        if (supported)
        {
            res = (*radioItf)->GetFreqRangeProperties(radioItf,
                    XA_FREQRANGE_FMJAPAN, &minFreq, &maxFreq,
                    &freqInterval); CheckErr(res);
            res = (*radioItf)->SetFreqRange(radioItf,
                    XA_FREQRANGE_FMJAPAN); CheckErr(res);
        }
        else
        {
            /* NO FM reception supported. Insert error message here. */
            exit(1);
        }
    }
    sem_wait(&semFreqRange);

    /* Start the playback */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_PLAYING); CheckErr(res);
}

/*
 * Shuts down the OpenMAX AL engine.
 */
void destroy (void)
{
    XAresult res;

    /* Stop the audio */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_STOPPED); CheckErr(res);

    /* Destroy the player */
    (*player)->Destroy(player);

    /* Destroy Output Mix object */
    (*outputMix)->Destroy(outputMix);
```

```
    /* Destroy Radio object */
    (*radio)->Destroy(radio);

    /* Shutdown OpenMAX AL */
    (*engine)->Destroy(engine);
}

/*
 * Called by UI when user tunes up or down.
 */
void tune (XAboolean tuneUp)
{
    XAresult res;
    XAuint32 newFreq;

    if (tuneUp == XA_BOOLEAN_TRUE)
    {
        newFreq = currentFreq + freqInterval;
        if (newFreq > maxFreq)
        {
            newFreq = minFreq;
        }
    }
    else
    {
        newFreq = currentFreq - freqInterval;
        if (newFreq < minFreq)
        {
            newFreq = maxFreq;
        }
    }

    res = (*radioItf)->SetFrequency(radioItf, newFreq); CheckErr(res);
    sem_wait(&semFreq);
    currentFreq = newFreq;

    /* Insert here code to update the new frequency to screen. */
}
```

D.4 Audio Recording through Microphone

```
/*
 *  OpenMAX AL - Audio Recording through Microphone Example
 */

#include <stdio.h>
#include <stdlib.h>

#include <OMXAL/OpenMAXAL.h>

#define MAX_NUMBER_INTERFACES    5
#define MAX_NUMBER_INPUT_DEVICES 3
#define POSITION_UPDATE_PERIOD   1000 /* 1 sec */

/* Checks for error. If any errors exit the application! */
```

```
void CheckErr (XAresult res)
{
    if (res != XA_RESULT_SUCCESS)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
}

void RecordEventCallback (
        XARecordItf caller,
        void *      pContext,
        XAuint32    recordevent)
{
    /* Callback code goes here */
}

/*
 * Test recording of audio from a microphone into a specified file
 */
void TestAudioRecording (XAObjectItf engine)
{
    XAObjectItf                   recorder;
    XARecordItf                   recordItf;
    XAEngineItf                   EngineItf;
    XAAudioIODeviceCapabilitiesItf AudioIODeviceCapabilitiesItf;
    XAAudioInputDescriptor        AudioInputDescriptor;
    XAresult                      res;

    XADataSource            audioSource;
    XADataLocator_IODevice  locator_mic;
    XADeviceVolumeItf       devicevolumeItf;

    XADataSink        audioSink;
    XADataLocator_URI uri;
    XADataFormat_MIME mime;

    int i;

    XAboolean     required[MAX_NUMBER_INTERFACES];
    XAInterfaceID iidArray[MAX_NUMBER_INTERFACES];

    XAuint32  InputDeviceIDs[MAX_NUMBER_INPUT_DEVICES];
    XAint32   numInputs     = 0;
    XAboolean mic_available = XA_BOOLEAN_FALSE;
    XAuint32  mic_deviceID  = 0;

    /* Get the XA Engine Interface, which is implicit */
    res = (*engine)->GetInterface(engine,
            XA_IID_ENGINE, (void*) &EngineItf); CheckErr(res);

    /* Get the Audio IO DEVICE CAPABILITIES interface, which is also
     * implicit */
    res = (*engine)->GetInterface(engine,
            XA_IID_AUDIOIODEVICECAPABILITIES,
            (void*) &AudioIODeviceCapabilitiesItf); CheckErr(res);
```

```
    numInputs = MAX_NUMBER_INPUT_DEVICES;

    res = (*AudioIODeviceCapabilitiesItf)->
        GetAvailableAudioInputs(AudioIODeviceCapabilitiesItf,
                &numInputs, InputDeviceIDs); CheckErr(res);

    /* Search for either earpiece microphone or headset microphone -
     * with a preference for the latter */
    for (i = 0; i < numInputs; i++)
    {
        res = (*AudioIODeviceCapabilitiesItf)->
            QueryAudioInputCapabilities(AudioIODeviceCapabilitiesItf,
                    InputDeviceIDs[i], &AudioInputDescriptor);
        CheckErr(res);
        if ((AudioInputDescriptor.deviceConnection ==
                    XA_DEVCONNECTION_ATTACHED_WIRED) &&
                (AudioInputDescriptor.deviceScope ==
                 XA_DEVSCOPE_USER) &&
                (AudioInputDescriptor.deviceLocation ==
                 XA_DEVLOCATION_HEADSET))
        {
            mic_deviceID  = InputDeviceIDs[i];
            mic_available = XA_BOOLEAN_TRUE;
            break;
        }
        else if ((AudioInputDescriptor.deviceConnection ==
                    XA_DEVCONNECTION_INTEGRATED) &&
                (AudioInputDescriptor.deviceScope ==
                 XA_DEVSCOPE_USER) &&
                (AudioInputDescriptor.deviceLocation ==
                 XA_DEVLOCATION_HANDSET))
        {
            mic_deviceID  = InputDeviceIDs[i];
            mic_available = XA_BOOLEAN_TRUE;
            break;
        }
    }

    /* If neither of the preferred input audio devices is available, no
     * point in continuing */
    if (!mic_available)
    {
        /* Appropriate error message here */
        exit(1);
    }

    /* Initialize arrays required[] and iidArray[] */
    for (i = 0; i < MAX_NUMBER_INTERFACES; i++)
    {
        required[i] = XA_BOOLEAN_FALSE;
        iidArray[i] = XA_IID_NULL;
    }

    /* Get the optional DEVICE VOLUME interface from the engine */
    res = (*engine)->GetInterface(engine,
```

```
        XA_IID_DEVICEVOLUME, (void*) &devicevolumeItf);
CheckErr(res);

/* Set recording volume of the microphone to -3 dB. This assumes that
   mic_device uses millibels; should use GetVolumeScale to be sure. */
res = (*devicevolumeItf)->SetVolume(devicevolumeItf,
        mic_deviceID, -300); CheckErr(res);

/* Setup the data source structure */
locator_mic.locatorType = XA_DATALOCATOR_IODEVICE;
locator_mic.deviceType  = XA_IODEVICE_AUDIOINPUT;
locator_mic.deviceID    = mic_deviceID;
locator_mic.device      = NULL;
audioSource.pLocator   = (void*) &locator_mic;
audioSource.pFormat    = NULL;

/* Setup the data sink structure */
uri.locatorType    = XA_DATALOCATOR_URI;
uri.pURI            = (XAchar *) "file:///recordsample.wav";
mime.formatType    = XA_DATAFORMAT_MIME;
mime.pMimeType      = (XAchar *) "audio/x-wav";
mime.containerType = XA_CONTAINERTYPE_WAV;
audioSink.pLocator = (void*) &uri;
audioSink.pFormat  = (void*) &mime;

/* Create media recorder with NULL for a the image/video source,
 * since this is for audio-only recording */
res = (*EngineItf)->CreateMediaRecorder(
        EngineItf, &recorder, &audioSource, NULL, &audioSink,
        0, iidArray, required); CheckErr(res);

/* Realizing the recorder in synchronous mode */
res = (*recorder)->Realize(recorder,
        XA_BOOLEAN_FALSE); CheckErr(res);

/* Get the RECORD interface - it is an implicit interface */
res = (*recorder)->GetInterface(recorder,
        XA_IID_RECORD, (void*) &recordItf); CheckErr(res);

/* Setup to receive position event callbacks */
res = (*recordItf)->RegisterCallback(recordItf,
        RecordEventCallback, NULL); CheckErr(res);

/* Set notifications to occur after every second - may be useful in
 * updating a recording progress bar */
res = (*recordItf)->SetPositionUpdatePeriod(recordItf,
        POSITION_UPDATE_PERIOD); CheckErr(res);
res = (*recordItf)->SetCallbackEventsMask(recordItf,
        XA_RECORDEVENT_HEADATNEWPOS); CheckErr(res);

/* Set the duration of the recording - 30 seconds (30,000
 * milliseconds) */
res = (*recordItf)->SetDurationLimit(recordItf,
        30000); CheckErr(res);

/* Record the audio */
```

```
    res = (*recordItf)->SetRecordState(recordItf,
            XA_RECORDSTATE_RECORDING);

    /* Destroy the recorder object */
    (*recorder)->Destroy(recorder);
}

int xa_main (void)
{
    XAresult    res;
    XAObjectItf engine;

    /* Create OpenMAX AL engine in thread-safe mode */

   XAEngineOption EngineOption[] = {
         (XAuint32) XA_ENGINEOPTION_THREADSAFE,
         (XAuint32) XA_BOOLEAN_TRUE,
         (XAuint32) XA_ENGINEOPTION_MAJORVERSION, (XAuint32) 1,
         (XAuint32) XA_ENGINEOPTION_MINORVERSION, (XAuint32) 1
   };

    res = xaCreateEngine(&engine,
            3, EngineOption, 0, NULL, NULL); CheckErr(res);

    /* Realizing the AL Engine in synchronous mode */
    res = (*engine)->Realize(engine, XA_BOOLEAN_FALSE); CheckErr(res);

    TestAudioRecording(engine);

    /* Shutdown OpenMAX AL engine */
    (*engine)->Destroy(engine);

    exit(0);
}
```

D.5 Snapshot with Preview

```
/*
 *  OpenMAX AL - Snapshot with Preview
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <OMXAL/OpenMAXAL.h>

#define MAX_NUMBER_INTERFACES 5


/* Dummy semaphore and event related types, prototypes and defines */

typedef XAuint16 Sem_t;  /* System semaphore type would replace Sem_t */

Sem_t semFocus;
```

```
Sem_t semCamInit;
Sem_t semCamShoot;

void sem_post (Sem_t * pSemaphore)
{
    /* Implementation specific semaphore post */
}

void sem_wait (Sem_t * pSemaphore)
{
    /* Implementation specific semaphore wait */
}

/* Checks for error. If any errors exit the application! */
void CheckErr (XAresult res)
{
    if (res != XA_RESULT_SUCCESS)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
}

/* CALLBACKS */

void CameraCallback (
        XACameraItf caller,
        void *      pContext,
        XAuint32    eventId,
        XAuint32    eventData)
{
    if (eventId == XA_CAMERACBEVENT_FOCUSSTATUS)
    {
        if (eventData == XA_CAMERA_FOCUSMODESTATUS_UNABLETOREACH)
        {
            /* Unable to focus to the subject now. Notify the user to
             * try again or shoot anyway. */
            /* This example just exits. */
            exit(1);
        }
        else if (eventData == XA_CAMERA_FOCUSMODESTATUS_REACHED)
        {
            sem_post(&semFocus);
        }
    }
}

void SnapshotInitiatedCallback (
        XASnapshotItf caller,
        void *        context)
{
    sem_post(&semCamInit);
}

void SnapshotTakenCallback (
        XASnapshotItf      caller,
```

```
        void *            context,
        XAuint32          numberOfPicsTaken,
        const XADataSink * image)
{
    if (numberOfPicsTaken != 1)
    {
        /* error: for some reason wrong number of pics were taken */
        exit(1);
    }
    sem_post(&semCamShoot);
}

void MetadataInsertionCallback (
    XAMetadataInsertionItf caller,
    void * pContext,
    XAMetadataInfo * pKey,
    XAMetadataInfo * pValue,
    XAint32 nodeID,
    XAboolean result)
{
    if(result != XA_RESULT_SUCCESS)
    {
         /* print an error message */
    }

    /* deallocate memory */
    free(pKey);
    free(pValue);

/* Please note that not all systems support deallocating memory in a
callback.  Please consult the target device's documentation.  An
alternative is to perform the memory deallocation in the application's
main thread and just notify the main thread from this callback. */
}

int main (int argc, char* argv[])
{
    XAObjectItf           engine;
    XAObjectItf           player;
    XAObjectItf           recorder;
    XAObjectItf           camera;
    XAPlayItf             playItf;
    XASnapshotItf         snapshotItf;
    XAMetadataInsertionItf metadataInsertionItf;
    XAImageEncoderItf     imageEncoderItf;
    XACameraItf           cameraItf;
    XAEngineItf           engineItf;

    /* camera */
    XADataSource          videoSource;
    XADataLocator_IODevice cameraLocator;

    /* viewfinder */
    XADataSink                  videoSink;
    XADataLocator_NativeDisplay displayLocator;
```

```
    /* In a real application, get the native handles from the operating
     * system. */
    XANativeHandle window = 0;
    XANativeHandle display = 0;

    /* output file */
    XADataSink        fileSink; /* output directory and picture name
                                   prefix */
    XADataLocator_URI uri;
    XADataFormat_MIME mime;


    XAresult res;

    XAImageSettings imageSettings;

    XAMetadataInfo * pKey;
    XAMetadataInfo * pValue;

    int i;

    XAboolean     required[MAX_NUMBER_INTERFACES];
    XAInterfaceID iidArray[MAX_NUMBER_INTERFACES];

    /* CREATE AL ENGINE */


   XAEngineOption EngineOption[] = {
         (XAuint32) XA_ENGINEOPTION_THREADSAFE,
         (XAuint32) XA_BOOLEAN_TRUE,
         (XAuint32) XA_ENGINEOPTION_MAJORVERSION, (XAuint32) 1,
         (XAuint32) XA_ENGINEOPTION_MINORVERSION, (XAuint32) 1
   };


    /* Initialize arrays required[] and iidArray[] */
    for (i = 0; i < MAX_NUMBER_INTERFACES; i++)
    {
        required[i] = XA_BOOLEAN_FALSE;
        iidArray[i] = XA_IID_NULL;
    }

    /* Create the OpenMAX AL engine */
    res = xaCreateEngine(&engine,
            3, EngineOption, 0, NULL, NULL); CheckErr(res);

    /* Realizing the Engine in synchronous mode. */
    res = (*engine)->Realize(engine, XA_BOOLEAN_FALSE); CheckErr(res);

    /*Get the XA Engine Interface which is implicit*/
    res = (*engine)->GetInterface(engine,
            XA_IID_ENGINE, (void*) &engineItf); CheckErr(res);

    /* CREATE CAMERA */

    /* Create Camera object (CameraItf is implicit) */
```

```
    res = (*engineItf)->CreateCameraDevice(engineItf,
            &camera, XA_DEFAULTDEVICEID_CAMERA, 0, NULL, NULL);
CheckErr(res);

    /* Realizing the Camera object synchronously. */
    res = (*camera)->Realize(camera, XA_BOOLEAN_FALSE); CheckErr(res);

    /* Get play and camera interfaces */
    res = (*camera)->GetInterface(camera,
            XA_IID_CAMERA, (void*) &cameraItf); CheckErr(res);

    /* Register callbacks */
    res = (*cameraItf)->RegisterCallback(cameraItf,
            CameraCallback, NULL); CheckErr(res);

    /* Setup the camera data source structure */
    cameraLocator.locatorType = XA_DATALOCATOR_IODEVICE;
    cameraLocator.deviceType  = XA_IODEVICE_CAMERA;
    cameraLocator.deviceID    = XA_DEFAULTDEVICEID_CAMERA;  /* ignored */
    cameraLocator.device      = camera;
    videoSource.pLocator      = (void*) &cameraLocator;
    videoSource.pFormat       = NULL;

    /* CREATE VIEWFINDER PLAYER */

    /* Setup the data sink structure for the viewfinder */
    displayLocator.locatorType = XA_DATALOCATOR_NATIVEDISPLAY;
    displayLocator.hWindow     = window;
    displayLocator.hDisplay    = display;
    videoSink.pLocator         = (void*) &displayLocator;
    videoSink.pFormat          = NULL;


    /* Create the player */
    res = (*engineItf)->CreateMediaPlayer(engineItf,
            &player, NULL, &videoSource, NULL, &videoSink, NULL, NULL,
            1, iidArray, required); CheckErr(res);

    /* Realizing the player in synchronous mode. */
    res = (*player)->Realize(player, XA_BOOLEAN_FALSE); CheckErr(res);

    /* Get the play interface */
    res = (*player)->GetInterface(player,
            XA_IID_PLAY, (void*) &playItf); CheckErr(res);

    /* start the viewfinder */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_PLAYING); CheckErr(res);

    /* CREATE RECORDER */

    /* Setup the data sink structure for the recorder */
    /* NULL since it is never used, but instead the DataSink is given in
     * InitiateSnapshot function */
    fileSink.pLocator = NULL;
    fileSink.pFormat  = NULL;
```

```
    /* Set arrays required[] and iidArray[] for SNAPSHOT, IMAGEENCODER
     * and METADATAINSERTION interfaces */
    required[0] = XA_BOOLEAN_TRUE;
    iidArray[0] = XA_IID_SNAPSHOT;
    required[1] = XA_BOOLEAN_TRUE;
    iidArray[1] = XA_IID_IMAGEENCODER;
    required[2] = XA_BOOLEAN_TRUE;
    iidArray[2] = XA_IID_METADATAINSERTION;

    /* Create the player */
    res = (*engineItf)->CreateMediaRecorder(engineItf,
            &recorder, NULL, &videoSource, &fileSink,
            3, iidArray, required); CheckErr(res);

    /* Realizing the player in synchronous mode. */
    res = (*recorder)->Realize(recorder,
            XA_BOOLEAN_FALSE); CheckErr(res);

    /* Get the interfaces */
    res = (*recorder)->GetInterface(recorder,
            XA_IID_SNAPSHOT, (void*) &snapshotItf); CheckErr(res);
    res = (*recorder)->GetInterface(recorder,
            XA_IID_IMAGEENCODER, (void*) &imageEncoderItf);
    CheckErr(res);
    res = (*recorder)->GetInterface(recorder,
            XA_IID_METADATAINSERTION, (void*) &metadataInsertionItf);
    CheckErr(res);

    /* Register callbacks */
    res = (*metadataInsertionItf)->RegisterCallback(metadataInsertionItf,
            MetadataInsertionCallback, NULL); CheckErr(res);

    /* SET-UP THE CAMERA */

    /* set the image codec and resolution - Real application should
     * first check the support with ImageEncoderCapabilitiesItf. */
    imageSettings.encoderId        = XA_IMAGECODEC_JPEG;
    imageSettings.width            = 1024;
    imageSettings.height           = 768;
    imageSettings.compressionLevel = 0; /* default compression */
    imageSettings.colorFormat      = 0; /* ignored (only used with raw
                                         * images) */
    res = (*imageEncoderItf)->SetImageSettings(imageEncoderItf,
            &imageSettings); CheckErr(res);

    /* free all automatic setting locks */
    res = (*cameraItf)->SetAutoLocks(cameraItf, 0); CheckErr(res);

    /* Assumes that all these modes are supported - Real application
     * should first check with CameraCapabilitiesItf. */
    res = (*cameraItf)->SetFocusMode(cameraItf,
            XA_CAMERA_FOCUSMODE_AUTO, 0, XA_BOOLEAN_TRUE); CheckErr(res);
    res = (*cameraItf)->SetExposureMode(cameraItf,
            XA_CAMERA_EXPOSUREMODE_AUTO, 0); CheckErr(res);
    res = (*cameraItf)->SetFlashMode(cameraItf,
```

```
        XA_CAMERA_FLASHMODE_AUTO); CheckErr(res);

/* Setup the file name prefix ("photo") and the path
 * ("C:/MyPictures/") */
uri.locatorType     = XA_DATALOCATOR_URI;
uri.pURI             = (XAchar *) "file:///C:/MyPictures/photo";
mime.formatType     = XA_DATAFORMAT_MIME;
mime.pMimeType       = (XAchar *) "image/jpeg";
mime.containerType  = XA_CONTAINERTYPE_UNSPECIFIED;
fileSink.pLocator   = (void*) &uri;
fileSink.pFormat    = (void*) &mime;

/* Setup the metadata to be inserted to the image file - Real
 * applications should use GetKeys to check the supported keys for
 * writing */

pKey = (XAMetadataInfo *) malloc(
        sizeof(XAMetadataInfo) + 12*sizeof(XAchar)); /* “KhronosTitle”
                                                        has 12 characters */
pKey->size = 13;
pKey->encoding = XA_CHARACTERENCODING_ASCII;
strcpy((char *)(pKey->langCountry), "en-us");
strcpy((char *)(pKey->data), "KhronosTitle");

pValue = (XAMetadataInfo *) malloc(
        sizeof(XAMetadataInfo) + 31*sizeof(XAchar));
pValue->size = 32;
pValue->encoding = XA_CHARACTERENCODING_ASCII;
strcpy((char *)(pValue->langCountry), "en-us");
strcpy((char *)(pValue->data),
        "This is the title of the image.");

res = (*metadataInsertionItf)->
    InsertMetadataItem(metadataInsertionItf,
            XA_ROOT_NODE_ID, pKey, pValue, XA_BOOLEAN_TRUE);
CheckErr(res);



/* Initiate shooting */
res = (*snapshotItf)->InitiateSnapshot(snapshotItf,
        1, 1, XA_BOOLEAN_TRUE, &fileSink, SnapshotInitiatedCallback,
        SnapshotTakenCallback, NULL); CheckErr(res);
sem_wait(&semCamInit);


/* FOCUSING (halfway press of the trigger) */

/* Lock the auto focus (Assumes that LOCK_AUTOFOCUS is supported -
 * Real application should check it first using
 * GetSupportedAutoLocks in XACameraCapabilitiesItf) */
res = (*cameraItf)->SetAutoLocks(cameraItf,
        XA_CAMERA_LOCK_AUTOFOCUS); CheckErr(res);

sem_wait(&semFocus);
```

```
    /* TAKE THE PICTURE */
    res = (*snapshotItf)->TakeSnapshot(snapshotItf); CheckErr(res);
    sem_wait(&semCamInit);

    /* The picture should be now in the directory specified by the
     * fileSink. */

    /* UNFREEZE THE VIEWFINDER */
    /* after the user has previewed the photo, unfreeze the viewfinder */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_PLAYING); CheckErr(res);

    /* SHUTDOWN */

    /* Stop the viewfinder */
    res = (*playItf)->SetPlayState(playItf,
            XA_PLAYSTATE_STOPPED); CheckErr(res);

    /* Destroy the objects */
    (*player)->Destroy(player);
    (*player)->Destroy(recorder);
    (*camera)->Destroy(camera);

    /* Shutdown OpenMAX AL */
    (*engine)->Destroy(engine);

    return 0;
}
```

D.6 Metadata Extraction

```
/*
 *  OpenMAX AL - Metadata Extraction
 */

#include <stdio.h>
#include <stdlib.h>

#include <OMXAL/OpenMAXAL.h>

/* Checks for error. If any errors exit the application! */
void CheckErr (XAresult res)
{
    if (res != XA_RESULT_SUCCESS)
    {
        /* Debug printing to be placed here */
        exit(1);
    }
}

/*
 * Prints all ASCII metadata key-value pairs (from the root of the media
 * since XAMetadataTraversalItf is not used)
 */
void TestMetadataSimple (XAMetadataExtractionItf mdExtrItf)
```

```
{
    XAresult res;
    XAuint32 mdCount = 0;
    XAuint32 i;

    /* scan through the metadata items */
    res = (*mdExtrItf)->GetItemCount(mdExtrItf, &mdCount); CheckErr(res);
    for (i = 0; i < mdCount; ++i)
    {
        XAMetadataInfo * key      = NULL;
        XAMetadataInfo * value    = NULL;
        XAuint32         itemSize = 0;

        /* get the size of and malloc memory for the metadata item */
        res = (*mdExtrItf)->GetKeySize(mdExtrItf,
                i, &itemSize); CheckErr(res);
        key = malloc(itemSize);
        if (key) /* no malloc error */
        {
            /* extract the key into the memory */
            res = (*mdExtrItf)->GetKey(mdExtrItf,
                    i, itemSize, key); CheckErr(res);
            if (key->encoding == XA_CHARACTERENCODING_ASCII)
            {
                res = (*mdExtrItf)->GetValueSize(mdExtrItf,
                        i, &itemSize); CheckErr(res);
                value = malloc(itemSize);
                if (value)  /* no malloc error */
                {
                    /* extract the value into the memory */
                    res = (*mdExtrItf)->GetValue(mdExtrItf,
                            i, itemSize, value); CheckErr(res);
                    if (value->encoding == XA_CHARACTERENCODING_ASCII)
                    {
                        printf("Item %d key: %s, value %s",
                                i, key->data, value->data);
                    }
                    free(value);
                }
            }
            free(key);
        }
    }
}
```