



# **OpenMAX™ Integration Layer Application Programming Interface Specification**

| Version 1.2.0 **Provisional**

Copyright © 2011 The Khronos Group Inc.

| November 7, 2011

Document version 1.2.0.0

Copyright © 2005-2011 The Khronos Group Inc. All Rights Reserved.

This specification is protected by copyright laws and contains material proprietary to the Khronos Group, Inc. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast, or otherwise exploited in any manner without the express prior written permission of the Khronos Group. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

Khronos Group grants express permission to any current Promoter, Contributor or Adopter member of Khronos to copy and redistribute UNMODIFIED versions of this specification in any fashion, provided that NO CHARGE is made for the specification and the latest available update of the specification for any version of the API is used whenever possible. Such distributed specification may be reformatted AS LONG AS the contents of the specification are not changed in any way. The specification may be incorporated into a product that is sold as long as such product includes significant independent work developed by the seller. A link to the current version of this specification on the Khronos Group website should be included whenever possible with specification distributions.

Khronos Group makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this specification, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose or non-infringement of any intellectual property. Khronos Group makes no, and expressly disclaims any, warranties, express or implied, regarding the correctness, accuracy, completeness, timeliness, and reliability of the specification. Under no circumstances will the Khronos Group, or any of its Promoters, Contributors or Members or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

SAMPLE CODE and EXAMPLES, as identified herein, are expressly depicted herein with a "grey" watermark and are included for illustrative purposes only and are expressly outside of the Scope as defined in Attachment A - Khronos Group Intellectual Property (IP) Rights Policy of the Khronos Group Membership Agreement. A Member or Promoter Member shall have no obligation to grant any licenses under any Necessary Patent Claims covering SAMPLE CODE and EXAMPLES.

Khronos and OpenMAX are trademarks of the Khronos Group Inc. Bluetooth is a registered trademark of the Bluetooth Special Interest Group. RealAudio and RealVideo are registered trademarks of RealNetworks, Inc. Windows Media is a registered trademark of Microsoft Corporation.

## Contents

<b>1 OVERVIEW .....</b>	<b>11</b>
1.1 INTRODUCTION.....	11
1.1.1 <i>About the Khronos Group</i> .....	11
1.1.2 <i>A Brief History of OpenMAX</i> .....	11
1.2 THE OPENMAX INTEGRATION LAYER .....	11
1.2.1 <i>Key Features and Benefits</i> .....	11
1.2.2 <i>Design Philosophy</i> .....	12
1.2.3 <i>Software Landscape</i> .....	13
1.2.4 <i>Stakeholders</i> .....	13
1.2.5 <i>The Interface</i> .....	14
1.3 DEFINITIONS.....	15
1.4 AUTHORS.....	16
1.5 FEATURES NEW TO VERSION 1.2 .....	17
1.6 BACKWARD COMPATIBILITY .....	20
<b>2 OPENMAX IL INTRODUCTION AND ARCHITECTURE.....</b>	<b>24</b>
2.1 ARCHITECTURAL OVERVIEW .....	24
2.2 KEY VOCABULARY .....	25
2.2.1 <i>Key Definitions</i> .....	26
2.3 SYSTEM COMPONENTS .....	27
2.3.1 <i>Component Profiles</i> .....	28
2.4 COMPONENT STATES.....	29
2.5 COMPONENT ARCHITECTURE.....	31
2.6 COMMUNICATION BEHAVIOR.....	32
2.7 THREAD SAFETY .....	33
2.8 TUNNELED BUFFER ALLOCATION.....	34
2.8.1 <i>IL Client Component Setup</i> .....	37
2.8.2 <i>Component Transition from OMX_StateLoaded to OMX_StateIdle</i> .....	37
2.9 PORT RECONNECTION .....	38
2.10 QUEUES AND FLUSH.....	39
2.11 MARKING BUFFERS .....	40
2.12 EVENTS AND CALLBACKS .....	40
2.13 BUFFER PAYLOAD .....	41
2.14 SIGNALLING FRAMES AND SUBFRAMES.....	44
2.14.1 <i>Signalling frames</i> .....	44
2.14.2 <i>Signalling subframes</i> .....	45
2.15 BUFFER FLAGS AND TIMESTAMP.....	47
2.16 SYNCHRONIZATION .....	47
2.17 RATE CONTROL .....	48
2.18 COMPONENT REGISTRATION .....	48
2.19 RESOURCE MANAGEMENT .....	48
2.19.1 <i>Need for Resource Management</i> .....	49
2.19.2 <i>Example Architecture</i> .....	50
2.19.3 <i>Component Priorities</i> .....	51
2.19.4 <i>Behavioral Rules</i> .....	51
2.19.5 <i>Hardware Vendor-Specific Resource Manager</i> .....	52
2.19.6 <i>Component Suspension</i> .....	52
2.20 CONTENT PIPES .....	53
2.21 FILE PARSING .....	53
2.22 VIDEO DECODER ERROR MAPPING .....	54
2.23 BUFFER PAYLOAD ADDITIONAL INFORMATION.....	54
2.23.1 <i>Buffer Data Formatting</i> .....	55
2.24 ENDIANNESS .....	56

<b>3 OPENMAX INTEGRATION LAYER CONTROL API .....</b>	<b>57</b>
3.1 OPENMAX IL TYPES .....	58
3.1.1 <i>Enumerations</i> .....	58
3.1.2 <i>Data Types</i> .....	75
3.1.3 <i>Structures</i> .....	77
3.2 OPENMAX IL CORE METHODS/MACROS.....	109
3.2.1 <i>Return Codes for the Functions</i> .....	110
3.2.2 <i>Macros</i> .....	112
3.2.3 <i>Functions</i> .....	141
3.3 OPENMAX IL COMPONENT METHODS AND STRUCTURES.....	153
3.3.1 <i>pComponentPrivate</i> .....	155
3.3.2 <i>pApplicationPrivate</i> .....	155
3.3.3 <i>GetComponentVersion</i> .....	155
3.3.4 <i>SendCommand</i> .....	155
3.3.5 <i>GetParameter</i> .....	156
3.3.6 <i>SetParameter</i> .....	156
3.3.7 <i>GetConfig</i> .....	156
3.3.8 <i>SetConfig</i> .....	156
3.3.9 <i>GetExtensionIndex</i> .....	157
3.3.10 <i>GetState</i> .....	157
3.3.11 <i>ComponentTunnelRequest</i> .....	157
3.3.12 <i>UseBuffer</i> .....	159
3.3.13 <i>AllocateBuffer</i> .....	159
3.3.14 <i>FreeBuffer</i> .....	159
3.3.15 <i>EmptyThisBuffer</i> .....	160
3.3.16 <i>FillThisBuffer</i> .....	160
3.3.17 <i>SetCallbacks</i> .....	160
3.3.18 <i>ComponentDeinit</i> .....	161
3.3.19 <i>UseEGLImage</i> .....	162
3.4 CALLING SEQUENCES.....	162
3.4.1 <i>Initialization</i> .....	162
3.4.2 <i>Data Flow</i> .....	171
3.4.3 <i>De-Initialization</i> .....	174
3.4.4 <i>Port Disablement and Enablement</i> .....	183
3.4.5 <i>Dynamic Port Reconfiguration</i> .....	187
3.4.6 <i>Autodetect Port Reconfiguration</i> .....	189
3.4.7 <i>Resource Management</i> .....	191
3.4.8 <i>Component Suspension</i> .....	195
3.5 SLAVING BEHAVIOR FOR PORT SETTINGS .....	199
<b>4 OPENMAX IL DATA API.....</b>	<b>200</b>
4.1 AUDIO.....	200
4.1.1 <i>Audio Use Case Examples</i> .....	200
4.1.2 <i>Minimum Buffer Payload Size for Uncompressed Data</i> .....	201
4.1.3 <i>Whole-file Buffering for MIDI Formats</i> .....	201
4.1.4 <i>General Enumerations</i> .....	201
4.1.5 <i>Parameter and Configuration Indexes</i> .....	204
4.1.6 <i>OMX_AUDIO_PORTDEFINITIONTYPE</i> .....	206
4.1.7 <i>OMX_AUDIO_PARAM_PORTFORMATTYPE</i> .....	207
4.1.8 <i>OMX_AUDIO_PARAM_PCMMODETYPE</i> .....	208
4.1.9 <i>OMX_AUDIO_PARAM_MP3TYPE</i> .....	210
4.1.10 <i>OMX_AUDIO_PARAM_AACPROFILETYPE</i> .....	212
4.1.11 <i>OMX_AUDIO_PARAM_VORBISTYPE</i> .....	215
4.1.12 <i>OMX_AUDIO_PARAM_WMATYPE</i> .....	216
4.1.13 <i>OMX_AUDIO_PARAM_RATYPE</i> .....	219

4.1.14	<i>OMX_AUDIO_PARAM_SBCTYPE</i> .....	220
4.1.15	<i>OMX_AUDIO_PARAM_ADPCMTYPE</i> .....	221
4.1.16	<i>OMX_AUDIO_PARAM_G723TYPE</i> .....	222
4.1.17	<i>OMX_AUDIO_PARAM_G726TYPE</i> .....	223
4.1.18	<i>OMX_AUDIO_PARAM_G729TYPE</i> .....	224
4.1.19	<i>OMX_AUDIO_PARAM_AMRTYPE</i> .....	225
4.1.20	<i>OMX_AUDIO_PARAM_GSMFRTYPE</i> .....	231
4.1.21	<i>OMX_AUDIO_PARAM_GSMEFRTYPE</i> .....	232
4.1.22	<i>OMX_AUDIO_PARAM_GSMHRTYPE</i> .....	233
4.1.23	<i>OMX_AUDIO_PARAM_TDMAFRTYPE</i> .....	233
4.1.24	<i>OMX_AUDIO_PARAM_TDMAEFRTYPE</i> .....	234
4.1.25	<i>OMX_AUDIO_PARAM_PDCFRTYPE</i> .....	235
4.1.26	<i>OMX_AUDIO_PARAM_PDCEFRTYPE</i> .....	236
4.1.27	<i>OMX_AUDIO_PARAM_PDCHRTYPE</i> .....	237
4.1.28	<i>OMX_AUDIO_PARAM_QCELP8TYPE</i> .....	238
4.1.29	<i>OMX_AUDIO_PARAM_QCELP13TYPE</i> .....	239
4.1.30	<i>OMX_AUDIO_PARAM_EVRCTYPE</i> .....	240
4.1.31	<i>OMX_AUDIO_PARAM_SMVTYPE</i> .....	242
4.1.32	<i>OMX_AUDIO_PARAM_MIDITYPE</i> .....	243
4.1.33	<i>OMX_AUDIO_PARAM_MIDILOADUSERSONDTYPE</i> .....	244
4.1.34	<i>OMX_AUDIO_CONFIG_MIDIIMMEDIATEEVENTTYPE</i> .....	246
4.1.35	<i>OMX_AUDIO_CONFIG_MIDISOUNDBANKPROGRAMTYPE</i> .....	247
4.1.36	<i>OMX_AUDIO_CONFIG_MIDICONTROLTYPE</i> .....	247
4.1.37	<i>OMX_AUDIO_CONFIG_MIDISTATUSTYPE</i> .....	249
4.1.38	<i>OMX_AUDIO_CONFIG_MIDIMETAEVENTTYPE</i> .....	251
4.1.39	<i>OMX_AUDIO_CONFIG_MIDIMETAEVENTDATATYPE</i> .....	252
4.1.40	<i>OMX_AUDIO_CONFIG_VOLUMETYPE</i> .....	252
4.1.41	<i>OMX_AUDIO_CONFIG_CHANNELVOLUMETYPE</i> .....	253
4.1.42	<i>OMX_AUDIO_CONFIG_BALANCETYPE</i> .....	254
4.1.43	<i>OMX_AUDIO_CONFIG_MUTETYPE</i> .....	254
4.1.44	<i>OMX_AUDIO_CONFIG_CHANNELMUTETYPE</i> .....	255
4.1.45	<i>OMX_AUDIO_CONFIG_LOUDNESSTYPE</i> .....	256
4.1.46	<i>OMX_AUDIO_CONFIG_BASSTYPE</i> .....	256
4.1.47	<i>OMX_AUDIO_CONFIG_TREBLETYPE</i> .....	257
4.1.48	<i>OMX_AUDIO_CONFIG_EQUALIZERTYPE</i> .....	257
4.1.49	<i>OMX_AUDIO_CONFIG_STEREOWIDENINGTYPE</i> .....	258
4.1.50	<i>OMX_AUDIO_CONFIG_CHORUSTYPE</i> .....	259
4.1.51	<i>OMX_AUDIO_CONFIG_REVERBERATIONTYPE</i> .....	260
4.1.52	<i>OMX_AUDIO_CONFIG_ECHOCANCELATIONTYPE</i> .....	262
4.1.53	<i>OMX_AUDIO_CONFIG_NOISEREDUCTIONTYPE</i> .....	263
4.1.54	<i>OMX_AUDIO_CONFIG_3DOUTPUTTYPE</i> .....	263
4.1.55	<i>OMX_AUDIO_CONFIG_3DLOCATIONTYPE</i> .....	264
4.1.56	<i>OMX_AUDIO_PARAM_3DDOPPLERMODETYPE</i> .....	265
4.1.57	<i>OMX_AUDIO_CONFIG_3DDOPPLERSETTINGSTYPE</i> .....	265
4.1.58	<i>OMX_AUDIO_CONFIG_3DLEVELSTYPE</i> .....	266
4.1.59	<i>OMX_AUDIO_CONFIG_3DDISTANCEATTENUATIONTYPE</i> .....	267
4.1.60	<i>OMX_AUDIO_CONFIG_3DDIRECTIVITYSETTINGSTYPE</i> .....	268
4.1.61	<i>OMX_AUDIO_CONFIG_3DDIRECTIVITYORIENTATIONTYPE</i> .....	269
4.1.62	<i>OMX_AUDIO_CONFIG_3DMACROSCOPICORIENTATIONTYPE</i> .....	269
4.1.63	<i>OMX_AUDIO_CONFIG_3DMACROSCOPICSIZETYPE</i> .....	270
4.1.64	<i>OMX_AUDIO_CHANNELMAPPINGTYPE</i> .....	271
4.1.65	<i>OMX_AUDIO_SBCBITPOOLTYPE</i> .....	272
4.1.66	<i>OMX_AUDIO_AMRMODETYPE</i> .....	272
4.1.67	<i>OMX_AUDIO_CONFIG_BITRATETYPE</i> .....	273
4.1.68	<i>OMX_AUDIO_CONFIG_AMRISFTYPE</i> .....	273
4.1.69	<i>OMX_AUDIO_FIXEDPOINTTYPE</i> .....	274

4.2 IMAGE AND VIDEO COMMON .....	275
4.2.1 <i>Uncompressed Data Formats</i> .....	276
4.2.2 <i>Minimum Buffer Payload Size for Uncompressed Data</i> .....	283
4.2.3 <i>Buffer Payload Requirements for Uncompressed Data</i> .....	283
4.2.4 <i>Parameter and Configuration Indexes</i> .....	284
4.2.5 <i>OMX_PARAM_DEBLOCKINGTYPE</i> .....	291
4.2.6 <i>OMX_PARAM_INTERLEAVETYPE</i> .....	291
4.2.7 <i>OMX_PARAM_SENSORMODETYPE</i> .....	292
4.2.8 <i>OMX_FRAMESETZETYPE</i> .....	293
4.2.9 <i>OMX_CONFIG_COLORCONVERSIONTYPE</i> .....	293
4.2.10 <i>OMX_CONFIG_SCALEFACTORTYPE</i> .....	295
4.2.11 <i>OMX_CONFIG_IMAGEFILTERTYPE</i> .....	295
4.2.12 <i>OMX_CONFIG_COLORENHANCEMENTTYPE</i> .....	296
4.2.13 <i>OMX_CONFIG_COLORKEYTYPE</i> .....	297
4.2.14 <i>OMX_CONFIG_COLORBLENDTYPE</i> .....	298
4.2.15 <i>OMX_CONFIG_FRAMESTABTYPE</i> .....	300
4.2.16 <i>OMX_CONFIG_ROTATIONTYPE</i> .....	300
4.2.17 <i>OMX_CONFIG_MIRRORTYPE</i> .....	301
4.2.18 <i>OMX_CONFIG_POINTTYPE</i> .....	302
4.2.19 <i>OMX_CONFIG_RECTTYPE</i> .....	302
4.2.20 <i>OMX_CONFIG_WHITEBALCONTROLTYP</i> E.....	303
4.2.21 <i>OMX_CONFIG_EXPOSURECONTROLTYP</i> E.....	304
4.2.22 <i>OMX_CONFIG_CONTRASTTYPE</i> .....	305
4.2.23 <i>OMX_CONFIG_BRIGHTNESSTYPE</i> .....	306
4.2.24 <i>OMX_CONFIG_BACKLIGHTTYPE</i> .....	306
4.2.25 <i>OMX_CONFIG_GAMMATYPE</i> .....	307
4.2.26 <i>OMX_CONFIG_SATURATIONTYPE</i> .....	307
4.2.27 <i>OMX_CONFIG_LIGHTNESSTYPE</i> .....	308
4.2.28 <i>OMX_CONFIG_PLANEBLENDTYPE</i> .....	308
4.2.29 <i>OMX_CONFIG_TRANSITIONEFFECTTYPE</i> .....	309
4.2.30 <i>OMX_CONFIG_DITHERTYPE</i> .....	310
4.2.31 <i>OMX_CONFIG_EXPOSUREVALUETYPE</i> .....	311
4.2.32 <i>OMX_OTHER_EXTRADATATYPE</i> .....	313
4.2.33 <i>OMX_CONFIG_CAPTUREMODETYPE</i> .....	316
4.2.34 <i>OMX_CONFIG_BOOLEANTYPE</i> .....	316
4.2.35 <i>OMX_SHARPNESSTYPE</i> .....	321
4.2.36 <i>OMX_CONFIG_ZOOMFACTORTYPE</i> .....	321
4.2.37 <i>OMX_IMAGE_CONFIG_LOCKTYPE</i> .....	323
4.2.38 <i>OMX_CONFIG_FOCUSRANGETYPE</i> .....	324
4.2.39 <i>OMX_IMAGE_CONFIG_FLASHSTATUSTYPE</i> .....	326
4.2.40 <i>OMX_CONFIG_EXTCAPTUREMODETYPE</i> .....	327
4.2.41 <i>OMX_CONFIG_NDFILTERCONTROLTYPE</i> .....	329
4.2.42 <i>OMX_CONFIG_AFASSISTANTLIGHTTYPE</i> .....	330
4.2.43 <i>OMX_FROITYPE</i> .....	338
4.2.44 <i>OMX_CONFIG_FOCUSREGIONSTATUSTYPE</i> .....	340
4.2.45 <i>OMX_MANUALFOCUSRECTTYPE</i> .....	341
4.2.46 <i>OMX_CONFIG_FOCUSREGIONCONTROLTYPE</i> .....	342
4.2.47 <i>OMX_INTERLACEFORMATTYPE</i> .....	343
4.2.48 <i>OMX_DEINTERLACETYPE</i> .....	346
4.2.49 <i>OMX_STREAMINTERLACEFORMATTYPE</i> .....	346
4.3 VIDEO.....	352
4.3.1 <i>Video Use Case Examples</i> .....	352
4.3.2 <i>General Enumerations</i> .....	353
4.3.3 <i>Parameter and Configuration Indices</i> .....	354
4.3.4 <i>OMX_VIDEO_PORTDEFINITIONTYPE</i> .....	356
4.3.5 <i>OMX_VIDEO_PARAM_PORTFORMATTYPE</i> .....	359

4.3.6	<i>OMX_VIDEO_PARAM_QUANTIZATIONTYPE</i> .....	360
4.3.7	<i>OMX_VIDEO_PARAM_VIDEOFASTUPDATETYPE</i> .....	361
4.3.8	<i>OMX_VIDEO_PARAM_BITRATETYPE</i> .....	362
4.3.9	<i>OMX_VIDEO_PARAM_MOTIONVECTORTYPE</i> .....	363
4.3.10	<i>OMX_VIDEO_PARAM_INTRAREFRESHTYPE</i> .....	364
4.3.11	<i>OMX_VIDEO_PARAM_ERRORCORRECTIONTYPE</i> .....	365
4.3.12	<i>OMX_VIDEO_PARAM_VBSMCTYPE</i> .....	366
4.3.13	<i>OMX_VIDEO_PARAM_H263TYPE</i> .....	367
4.3.14	<i>OMX_VIDEO_PARAM_MPEG2TYPE</i> .....	369
4.3.15	<i>OMX_VIDEO_PARAM_MPEG4TYPE</i> .....	371
4.3.16	<i>OMX_VIDEO_PARAM_WMVTYPE</i> .....	374
4.3.17	<i>OMX_VIDEO_PARAM_RVTYPE</i> .....	375
4.3.18	<i>OMX_VIDEO_PARAM_AVCTYPE</i> .....	376
4.3.19	<i>OMX_VIDEO_PARAM_VP8TYPE</i> .....	379
4.3.20	<i>OMX_VIDEO_VP8REFERENCEFRAMETYPE</i> .....	381
4.3.21	<i>OMX_VIDEO_VP8REFERENCEFRAMEINFOTYPE</i> .....	383
4.3.22	<i>OMX_VIDEO_CONFIG_BITRATETYPE</i> .....	384
4.3.23	<i>OMX_CONFIG_FRAMERATETYPE</i> .....	384
4.3.24	<i>OMX_CONFIG_INTRAREFRESHVOPTYPE</i> .....	385
4.3.25	<i>OMX_CONFIG_MACROBLOCKERRORMAPTYPE</i> .....	385
4.3.26	<i>OMX_PARAM_MACROBLOCKSTYPE</i> .....	386
4.3.27	<i>OMX_CONFIG_MBERRORREPORTINGTYPE</i> .....	387
4.3.28	<i>OMX_VIDEO_PARAM_PROFILELEVELTYPE</i> .....	388
4.3.29	<i>OMX_VIDEO_PARAM_AVCSLICEFMO</i> .....	391
4.3.30	<i>OMX_VIDEO_CONFIG_AVCINTRAPERIOD</i> .....	393
4.3.31	<i>OMX_VIDEO_CONFIG_NALSIZE</i> .....	393
4.3.32	<i>OMX_NALSTREAMFORMATTYPE</i> .....	394
4.3.33	<i>OMX_VIDEO_PARAM_VC1TYPE</i> .....	399
4.3.34	<i>OMX_VIDEO_INTRAPERIODTYPE</i> .....	401
4.4	<b>IMAGE</b> .....	402
4.4.1	<i>Image Use Case Example</i> .....	403
4.4.2	<i>Parameter and Configuration Indices</i> .....	406
4.4.3	<i>OMX_IMAGE_PORTDEFINITIONTYPE</i> .....	406
4.4.4	<i>OMX_IMAGE_PARAM_PORTFORMATTYPE</i> .....	409
4.4.5	<i>OMX_IMAGE_PARAM_FLASHCONTROLTYPE</i> .....	409
4.4.6	<i>OMX_IMAGE_CONFIG_FOCUSCONTROLTYPE</i> .....	410
4.4.7	<i>OMX_IMAGE_PARAM_QFACTORTYPE</i> .....	412
4.4.8	<i>OMX_IMAGE_PARAM_QUANTIZATIONTABLETYPE</i> .....	412
4.4.9	<i>OMX_IMAGE_PARAM_HUFFMANTABLETYPE</i> .....	413
4.4.10	<i>OMX_CONFIG_FLICKERREJECTIONTYPE</i> .....	415
4.4.11	<i>OMX_IMAGE_HISTOGRAMTYPE</i> .....	415
4.4.12	<i>OMX_IMAGE_HISTOGRAMDATATYPE</i> .....	416
4.4.13	<i>OMX_IMAGE_HISTOGRAMINFOTYPE</i> .....	417
4.4.14	<i>OMX_CONFIG_FILEFORMATTYPE</i> .....	418
4.4.15	<i>IMAGE CAPTURE START-END NOTIFICATIONS</i> .....	419
4.5	<b>“OTHER” DOMAIN</b> .....	420
4.5.1	<i>Parameters and Config Indexes</i> .....	420
4.5.2	<i>OMX_TIME_CONFIG_SEEKMODETYPE</i> .....	421
4.5.3	<i>OMX_TIME_CONFIG_TIMESTAMPTYPE</i> .....	422
4.5.4	<i>OMX_TIME_CONFIG_MEDIATIMEREQUESTTYPE</i> .....	423
4.5.5	<i>OMX_TIME_CONFIG_MEDIATIMETYPE</i> .....	423
4.5.6	<i>OMX_TIME_CONFIG_SCALETYPE</i> .....	425
4.5.7	<i>OMX_TIME_CONFIG_CLOCKSTATETYPE</i> .....	425
4.6	<b>COMMON OR DOMAIN INDEPENDENT</b> .....	427
4.6.1	<i>Parameter and Configuration Indices</i> .....	427
4.6.2	<i>OMX_CONFIG_COMMITMODETYPE</i> .....	428

4.6.3	<i>OMX_CONFIG_COMMITTYPE</i>	429
4.6.4	<i>OMX_CONFIG_CALLBACKREQUESTTYPE</i>	430
4.6.5	<i>OMX_MEDIACONTAINER_INFOTYPE</i>	430
4.6.6	<i>OMX_CONFIG_PORTBOOLEANTYPE</i>	432
<b>5</b>	<b>OPENMAX IL COMPONENT EXTENSION APIs</b>	<b>433</b>
5.1	DESCRIPTION OF THE EXTENSION PROCESS	433
5.1.1	<i>GetExtensionIndex</i>	434
5.1.2	<i>Custom Data Structures</i>	435
5.1.3	<i>Enumerations</i>	435
5.1.4	<i>Promoting extensions to specification</i>	435
5.2	EXAMPLES OF USING EXTENSION QUERYING API	435
5.2.1	<i>Sample Code Showing Calling Sequence</i>	436
<b>6</b>	<b>SYNCHRONIZATION</b>	<b>437</b>
6.1	SEEKING COMPONENT	437
6.1.1	<i>Seeking Configurations</i>	437
6.1.2	<i>Seeking Buffer Flags</i>	438
6.1.3	<i>Seek Event Sequence</i>	438
6.2	CLOCK COMPONENT	439
6.2.1	<i>Timestamps</i>	439
6.2.2	<i>Media Clock</i>	440
6.2.3	<i>Wall Clock</i>	442
6.2.4	<i>Reference Clocks</i>	442
6.2.5	<i>Rendering Delay</i>	448
6.2.6	<i>Clock Component Implementation</i>	450
6.2.7	<i>Audio-Video File Playback Example Use Case</i>	453
<b>7</b>	<b>CONTAINER PARSING</b>	<b>456</b>
7.1	PARAMETER AND CONFIGURATION INDEXES	456
7.2	FORMAT DETECTION	457
7.3	PORT STREAMS	457
7.4	METADATA EXTRACTION	458
7.5	TYPES AND STRUCTURES	460
7.5.1	<i>OMX_PARAM_U32TYPE</i>	460
7.5.2	<i>OMX_METADATACHARSETTYPE</i>	460
7.5.3	<i>OMX_METADATASCOPETYPE</i>	462
7.5.4	<i>OMX_CONFIG_METADATAITEMCOUNTTYPE</i>	462
7.5.5	<i>OMX_CONFIG_METADATAITEMTYPE</i>	464
7.5.6	<i>OMX_PARAM_METADATAFILTERTYPE</i>	466
7.5.7	<i>OMX_CONFIG_CONTAINERNODECOUNTTYPE</i>	469
7.5.8	<i>OMX_CONFIG_CONTAINERNODEIDTYPE</i>	469
<b>8</b>	<b>MANDATORY COMPONENT PARAMETERS</b>	<b>471</b>
8.1	COMPONENT ROLE	471
8.1.1	<i>ComponentRoleEnum</i>	471
8.1.2	<i>OMX_PARAM_COMPONENTROLETYPE</i>	472
8.1.3	<i>OMX_RoleOfComponentEnum</i>	472
8.1.4	<i>OMX_ComponentOfRoleEnum</i>	473
8.2	MANDATORY PORT PARAMETERS	473
<b>9</b>	<b>STANDARD COMPONENTS</b>	<b>475</b>
9.1	HIERARCHY OF STANDARD COMPONENT DEFINITION	475
9.1.1	<i>Standard Component Class Definition</i>	476
9.1.2	<i>Standard Components Definition</i>	476
9.2	NOTATION USED	480

9.3	VIDEO AND IMAGE ORDER OF OPERATIONS .....	480
9.4	STANDARD AUDIO COMPONENTS .....	481
9.4.1	<i>Audio Decoder Class</i> .....	481
9.4.2	<i>Audio Encoder Class</i> .....	494
9.4.3	<i>Audio Mixer Class</i> .....	506
9.4.4	<i>Audio Reader Class</i> .....	509
9.4.5	<i>Audio Renderer Class</i> .....	509
9.4.6	<i>Audio Writer Class</i> .....	511
9.4.7	<i>Audio Capturer Class</i> .....	511
9.4.8	<i>Audio Processor class</i> .....	514
9.4.9	<i>Audio 3D Mixer Class</i> .....	519
9.5	STANDARD IMAGE COMPONENTS .....	526
9.5.1	<i>Image Decoder Class</i> .....	526
9.5.2	<i>Image Encoder Class</i> .....	530
9.5.3	<i>Image Reader Class</i> .....	534
9.5.4	<i>Image Writer Class</i> .....	534
9.6	STANDARD VIDEO COMPONENTS .....	534
9.6.1	<i>Video Decoder Class</i> .....	534
9.6.2	<i>Video Encoder Class</i> .....	549
9.6.3	<i>Video Reader Class</i> .....	560
9.6.4	<i>Video Scheduler Class</i> .....	561
9.6.5	<i>Video Writer Class</i> .....	561
9.7	OTHER STANDARD COMPONENTS .....	562
9.7.1	<i>Camera Class</i> .....	562
9.7.2	<i>Clock Class</i> .....	571
9.7.3	<i>Container Demuxer Class</i> .....	572
9.7.4	<i>Container Muxer Class</i> .....	574
9.7.5	<i>Image/Video Processor Class</i> .....	575
9.7.6	<i>Image/Video Renderer Class</i> .....	578
10	IMPLEMENTING BUFFER SHARING .....	599
11	APPENDIX A – REFERENCES .....	604
11.1	SPEECH .....	604
11.1.1	<i>3GPP</i> .....	604
11.1.2	<i>3GPP2</i> .....	604
11.1.3	<i>ARIB</i> .....	604
11.1.4	<i>ITU</i> .....	604
11.1.5	<i>IETF</i> .....	605
11.1.6	<i>TIA</i> .....	605
11.2	AUDIO .....	605
11.2.1	<i>ISO</i> .....	605
11.2.2	<i>MISC</i> .....	606
11.3	SYNTHETIC AUDIO .....	606
11.3.1	<i>MIDI</i> .....	606
11.4	IMAGE .....	607
11.4.1	<i>IETF</i> .....	607
11.4.2	<i>ISO</i> .....	608
11.4.3	<i>ITU</i> .....	608
11.4.4	<i>JEITA</i> .....	609
11.4.5	<i>MIPI</i> .....	609
11.4.6	<i>Miscellaneous</i> .....	609
11.4.7	<i>SMIA</i> .....	609
11.4.8	<i>W3C</i> .....	610
11.5	VIDEO .....	610
11.5.1	<i>3GPP</i> .....	610

<i>11.5.2</i>	<i>AVS.....</i>	<i>610</i>
<i>11.5.3</i>	<i>DLNA.....</i>	<i>610</i>
<i>11.5.4</i>	<i>ETSI.....</i>	<i>610</i>
<i>11.5.5</i>	<i>IETF.....</i>	<i>611</i>
<i>11.5.6</i>	<i>ISO.....</i>	<i>611</i>
<i>11.5.7</i>	<i>ITU.....</i>	<i>612</i>
<i>11.5.8</i>	<i>MISC.....</i>	<i>612</i>
<i>11.6</i>	<i>JAVA .....</i>	<i>612</i>
<i>11.6.1</i>	<i>Multimedia.....</i>	<i>612</i>
<i>11.6.2</i>	<i>Broadcast.....</i>	<i>612</i>

# 1 Overview

## 1.1 Introduction

This document details the Application Programming Interface (API) for the OpenMAX Integration Layer (IL). Developed as an open standard by The Khronos Group, the IL serves as a low-level interface for audio, video, and imaging components used in embedded and/or mobile devices. The principal goal of the IL is to give components a degree of system abstraction for the purpose of portability across operating systems and software stacks.

### 1.1.1 *About the Khronos Group*

The Khronos Group is a member-funded industry consortium focused on the creation of open standard APIs to enable the authoring and playback of dynamic media on a wide variety of platforms and devices. All Khronos members may contribute to the development of Khronos API specifications, may 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, OpenML, and OpenVG.

### 1.1.2 *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 stretching the breadth 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 IL API makes it applicable to all media platforms.

## 1.2 The OpenMAX Integration Layer

The OpenMAX IL API strives to give media components portability across an array of platforms. The interface abstracts the hardware and software architecture in the system. Each component and relevant transform is encapsulated in a component interface. The OpenMAX IL API allows the user to load, control, connect, and unload the individual components. This flexible core architecture allows the Integration Layer to easily implement almost any media use case and mesh with existing graph-based media frameworks.

### 1.2.1 *Key Features and Benefits*

The OpenMAX IL API gives applications and media frameworks the ability to interface with multimedia codecs and supporting components (i.e., sources and sinks) in a unified

manner. The components themselves may be any combination of hardware or software and are completely transparent to the user. Without a standardized interface of this nature, component vendors have little alternative than to write to proprietary or closed interfaces to integrate into mobile devices. In this case, the portability of the component is minimal at best, costing many development-years of effort in re-tooling these solutions between systems.

Thus, the IL incorporates a specialized arsenal of features, honed to combat the problem of portability among many vastly different media systems. Such features include:

- A flexible component-based API core
- Ability to easily plug in new components
- Coverage of targeted domains (audio, video, and imaging) while remaining easily extensible by both the Khronos Group and individual vendors
- Capable of being implemented as either static or dynamic libraries
- Retention of key features and configuration options needed by parent software (such as media frameworks)
- Ease of communication between the client and the components and between components themselves
- Standardized definition of key components so all implementations of such “standard components” expose the same external interface (i.e. same inputs, outputs, and controls)

### **1.2.2     *Design Philosophy***

As previously stated, the key focus of the OpenMAX IL API is portability of media components. The diversity of existing devices and media implementation solutions necessitates that the OpenMAX IL target the higher level of the media software stack as the key initial user. For many operating systems, this means an existing media framework or some form of multimedia middleware.

Another key target is the OpenMAX AL API which standardizes a higher application level interface companion to OpenMAX IL. OpenMAX AL is designed to be amenable to OpenMAX IL implementations.

Thus, much of the OpenMAX IL API accommodates the needs of multimedia middleware allowing that layer to be as lightweight as possible. The result is an interface that is easily pluggable into most software stacks across operating system and multimedia middleware solutions.

The design of the API also strove to accommodate as many system architectures as possible. The resulting design uses highly asynchronous communications, which allows processing to take place in another thread, on multiple processing elements, or on specialized hardware. In addition, the ability of hardware-accelerated components to communicate directly with one another via tunneling affords implementation architectures even greater flexibility and efficiency.

### 1.2.3 Software Landscape

In some systems, a user-level media framework already exists. In those without such multimedia middleware, OpenMAX AL may serve to fill the gap. The OpenMAX IL API is designed to easily fit below this layer with little to no overhead between the interfaces. In most cases, a native media framework can be replaced with a thin layer that simply translates the API. Likewise, given the co-operative design of the two APIs, OpenMAX IL can even more seamlessly fit into an OpenMAX AL implementation. Figure 1-1 illustrates the software landscape for the OpenMAX IL API.

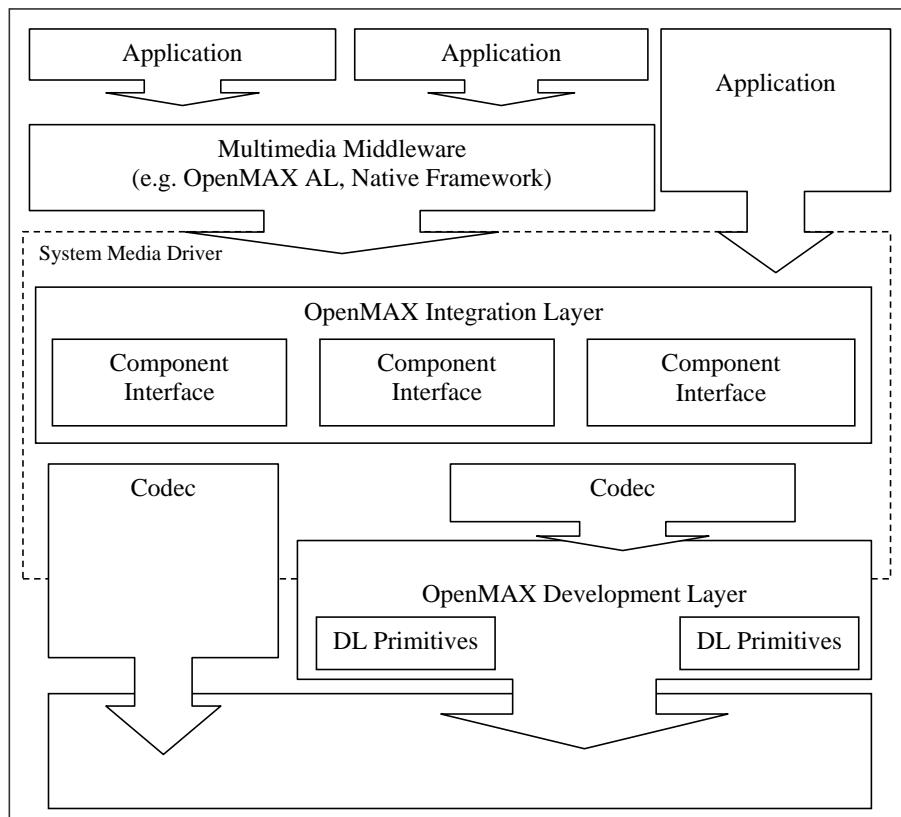


Figure 1-1. OpenMAX IL API Software Landscape

The OpenMAX standard also defines a set of Development Layer (DL) primitives on which components can be built. The DL primitives and their full relationship to the IL are specified in the OpenMAX Development Layer API specification documents.

### 1.2.4 Stakeholders

A few categories of stakeholders represent the broad array of companies participating in the production of multimedia solutions, each with their own interest in the IL API.

#### **1.2.4.1 Silicon Vendors**

Silicon vendors (SV) are responsible for delivering a representative set of OpenMAX IL components that are specific to the vendor's platform. The vendors are anticipated to also supply components that are representative of the capabilities of their platforms.

#### **1.2.4.2 Independent Software Vendors**

Independent software vendors (ISV) are anticipated to deliver additional differentiated OpenMAX IL components that may or may not be specific to a given silicon vendor's platform.

#### **1.2.4.3 Operating System Vendors**

Operating System Vendors (OSV) are anticipated to deliver software multimedia framework and standard reference OpenMAX IL components that enable integration of the representative silicon vendor's components and ISV components. ~~The OSV is responsible for conformance testing of the standard reference OpenMAX IL components.~~

#### **1.2.4.4 Original Equipment Manufacturers**

Original Equipment Manufacturers (OEM) are anticipated to modify and optimize the integration of OpenMAX IL components provided by SVs, ISVs, and OSVs to their specific product architectures to enable delivery of OpenMAX IL integrated multimedia devices. OEMs may also develop and integrate their own proprietary OpenMAX IL components.

### **1.2.5 The Interface**

The OpenMAX IL API is a component-based media API that consists of two main segments: the core API and the component API.

#### **1.2.5.1 Core**

The OpenMAX IL core is used for dynamically loading and unloading components and for facilitating component communication. Once loaded, the API allows the user to communicate directly with the component, which eliminates any overhead for high level commands. Similarly, the core allows a user to establish a communication tunnel between two components. Once established, the core API is no longer used and communications flow directly between components.

#### **1.2.5.2 Components**

In the OpenMAX Integration Layer, components represent individual blocks of functionality. Components can be sources, sinks, codecs, filters, splitters, mixers, or any other data operator. Depending on the implementation, a component could possibly represent a piece of hardware, a software codec, another processor, or a combination thereof.

The individual parameters of a component can be set or retrieved through a set of associated data structures, enumerations, and interfaces. The parameters include data relevant to the component's operation (i.e., codec options) or the actual execution state of the component.

Buffer status, errors, and other time-sensitive data are relayed to the application via a set of callback functions. These are set via the normal parameter facilities and allow the API to expose more of the asynchronous nature of system architectures.

Data communication to and from a component is conducted through interfaces called ports. Ports represent both the connection for components to the data stream and the buffers needed to maintain the connection. Users may send data to components through input ports or receive data through output ports. Similarly, a communication tunnel between two components can be established by connecting the output port of one component to a similarly formatted input port of another component.

### 1.3 Definitions

When this specification discusses requirements and features of the OpenMAX IL API, specific words are used to convey their necessity in an implementation. Table 1-1 shows a list of these words.

**Table 1-1: Definitions of Commonly Used Words**

Word	Definition
May	The stated functionality is an optional requirement for an implementation of the OpenMAX IL 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 component may have vendor specific extensions."
Shall	The stated functionality is a requirement for an implementation of the OpenMAX IL API. If a component fails to meet a shall statement, it is not considered to conform to this specification. Shall is always used as a requirement, as in "The component designers shall produce good documentation."
Should	The stated functionality is not a requirement for an implementation of the OpenMAX IL API but is recommended or is a good practice. Should is usually used as follows: "The component should begin processing buffers immediately after it transitions to the OMX_StateExecuting state." While this is good practice, there may be a valid reason to delay processing buffers, such as not having input data available.
Will	The stated functionality is not a requirement for an implementation of the OpenMAX IL API. Will is usually used when referring to a third party, as in "the application framework will correctly handle errors."

## 1.4 Authors

The following individuals, listed alphabetically by company, contributed to this OpenMAX Integration Layer Application Programming Interface Specification.

- [Tom Longo \(AMD\)](#)
- [Wilson Kwan \(AMD\)](#)
- [Russell Tillitt \(Beatnik\)](#)
- Tim Granger (Broadcom)
- [Roger Nixon \(Broadcom\)](#)
- [Harald Gustafson \(Ericsson\)](#)
- [Zhengrong Yao \(Ericsson\)](#)
- [Juha Ylinen \(Google\)](#)
- [Glenn Kasten \(Google\)](#)
- [Kevan Ahmadi \(Imagination Technologies\)](#)
- [Neeraj Agrawal \(Imagination Technologies\)](#)
- [Dave Murray \(Incoras\)](#)
- [Adrian Burian \(Nokia\)](#)
- [Brian Evans \(Nokia\)](#)
- [Jarmo Hiipakka \(Nokia\)](#)
- [Juan Rubio \(Nokia\)](#)
- [Maria Pascual-Borrego \(Nokia\)](#)
- [Rajesh Rathinasamy \(Nokia\)](#)
- [Sriram Divakar \(Motorola\)](#)
- [Yeshwant Muthusamy \(Nokia\)](#)
- [Isaac Richards \(NVIDIA\)](#)
- [Ukri Niemimäkki \(Nokia\)](#)
- [Gordon Grigor \(NVIDIA\)](#)
- [Jim Van Welzen \(NVIDIA\)](#)
- [Dusan Veselinovic \(PacketVideo\)](#)
- [Bruno Smets \(NXP\)](#)
- [Tom Longo \(Qualcomm\)](#)
- [Alwyn Dos Remedios \(Qualcomm\)](#)

- [Eric Auger \(ST-Ericsson\)](#)
- [Laurent Gerard \(ST-Ericsson\)](#)
- [Sebastien Le Duc \(ST-Ericsson\)](#)
- [Thierry Vuillaume \(ST-Ericsson\)](#)
- [Diego Melpignano \(STMicroelectronics\)](#)[Giulio Urlini \(STMicroelectronics\)](#)
- [Sripal Bagadia \(Texas Instruments\)](#)
- [Nikhil Mande \(Texas Instruments\)](#)
- [Leo Estevez \(Texas Instruments\)](#)

## 1.5 Features New to Version 1.24

A summary of new features included into this release of this specification include:

### State Machine Related Updates

- [Introduction of dynamically allocated buffer support in addition to statically allocating buffers](#)
- [Removal of OMX\\_StateInvalid state](#)
- [Introduction of tunneled port status in order to eliminate race conditions during state transitions\port enabled when attempting to accept buffer usage and initiate buffer exchanges](#)
- [Ability to cancel pending submitted commands in order to recover from dead-lock situations](#)
- [Clarification of flush operation behavior](#)

### Buffer Flags Updates

- [Ability to signal sub-frame boundaries](#)
- [Ability to signal the presence of valid timestamp information](#)
- [Ability to signal read-only buffers](#)

### Events Updates

- [Updated EventHandler parameter usage to clarify event reasons](#)
- [New Event Types](#)
  - [Ability to detect the need to disable or flush other component ports in cases of buffer sharing](#)
  - [Ability to detect parameter or config index settings](#)
- [Updated conditions for when Port Settings Changes are to be signaled](#)

### Error Codes Updates

- Clean up of error code types: deletion of types, addition of types and updated descriptions for some existing types
- Updated cross references between methods and the errors that may be reported.

#### Methods Updates

- Introduction of separate OMX IL core methods to handle component ports tunnel setup and tear down
- Ability to introduce OMX IL core related extensions
- Ability to update the component callback method

#### Audio Features Updates

- Introduction of 3D Audio support
- Introduction of AMR WB+ support
- Extended WMA and AMR formats
- Ability to dynamically update settings such as bitrate, AMR modes and SBC Bitpool sizes

#### Camera Features Updates

- Enhanced Digital and Optical Zoom support
- Enhanced Exposure, White Balance and Focus Lock support
- Enhanced capture mode settings: pre-capture enablement and per port capturing control
- Enhanced Focus Range, Region and Status support
- Introduction of Field of View controls
- Introduction of Flash status reporting
- Introduction of ND Filter support
- Introduction of Assistant Light Control support
- Information of Flicker Rejection support
- Introduction of Histogram information
- Introduction of Sharpness control
- Ability to synchronize shutter opening and closing events with audio playback

#### Video Features Updates

- Introduction of Interlace detection of processing support
- Introduction of VC1 support
- Introduction of VP8 support
- Introduction of NAL Format support

#### Imaging Features Updates

- Introduction of additional Image Filters

#### Clock Features Updates

- Enhanced reference clock selections
- Enhanced media time notification mechanism

#### Standard Components Updates

- Introduction of a new header file to encapsulate all the standard component role names
- Updated feature list and processing criteria: e.g. additional ports, expanded list of supportable color formats, port slaving behavior, mandating that audio renders shall support the ability to provide reference clocks
- New standard components: AMR-WB+ Decoder\Encoder, 3D Audio Mixers, VC1 Video Decoder\Encoder

#### Other Updates

- Ability to group and commit multiple configuration settings atomically
- Clarifications of various parameter usage and processing operations
- Additional color formats
- The explicit definition of a set of standard components representing the most common pieces of functionality (e.g. specific data sources, sinks, decoders, encoders, transformations for specific formats). All implementations of particular standard component expose the same interface including inputs, outputs, and controls.
- The addition of the ability to append additional information to buffer payloads (e.g. the video quantization data appended to video frames).
- The extension of color format types.
- The extension of buffer payload flags
- The clarification of data transform (e.g. rotate and scale) order
- The introduction media container parsing and creating, including the abstraction of file access (denoted content pipes) and metadata parsing.
- The enhancement of Resource Management to include suspension due to unavailable resources (a lightweight alternative to component deinitialization on resource loss), resource Concealment control, dynamic resource allocation.
- The addition of a means to specify an EGL buffer to be used as an OpenMAX IL buffer.
- The addition of MP3 file formats.

- The enhancement of video encoder controls including dynamic frame rate and bit rate, intraframe and macroblock refresh, FMO and Slice selection, IDR and intra period selection, NAL size selection, video profile querying.
- The enhancement of video decoder controls including macroblock error reporting during decoding and video profile querying.
- The enhancement of image codec controls including more sophisticated controls for JPEG Huffman and Quantization tables
- The enhancement of camera controls including sophisticated focus control, continuous and single shot control, auto exposure control.

## 1.6 Backward Compatibility

The OpenMAX IL specification defines components and structures that evolve and improve with subsequent versions of the specification. The version of the specification is indicated with 4 digits Ma.Mi.R.S (Respectively Major, Minor, Revision and Step). Increments of these digits give the following indications:

- An increment of Major indicates a significant number of fundamental non-backward compatible changes.
- An increment of Minor indicates a significant number of functional changes like the addition of new structures and components. Essential corrections may create limited non backward compatible changes. Heterogeneous Minor version implementations should be possible as explained below for 1.0 to 1.1.
- An increment of revision indicates a significant number of corrections and clarifications which should be backward compatible unless stated explicitly. Any component of a later revision should interoperate with components of an earlier revision.
- An increment of step indicates a significant number of editorial corrections.

This specification version continues to support a significant level of functionality available as part of previous releases, however due to the nature of some of the improvements introduced, backwards compatibility with previous versions is not being maintained.

OpenMAX IL cCore and component providers shall only be required to provide functionality as described in this specification.

While backwards compatibility is not a requirement of the specification, OpenMAX IL solution providers may choose to include support for previous specification releases as part of their offerings works to maintain backward compatibility with the OpenMAX Integration Layer Specification 1.0 to aid in the adoption and deployment of the specification.

It is recognized that systems of heterogeneous pieces from prior versions and this version of the specification may exist. As such, new features and modifications to existing

~~features part of this specification need to provide a standardized mechanism for backward compatibility. Thus systems with heterogeneous OpenMAX IL clients, IL core, and IL components can operate together.~~

~~Backward compatibility is required on any interfaces where an OpenMAX IL 1.0 piece (i.e. a client, core, or component) connects to an OpenMAX IL 1.1 piece. The cost of backward compatibility is placed on the OpenMAX IL 1.1 pieces. So OpenMAX IL 1.1 components and OpenMAX IL 1.1 cores shall support backward compatibility with OpenMAX IL 1.0.~~

~~The specification enables OpenMAX IL 1.1 pieces to support backward compatibility.~~

~~This is achieved by providing structures which maintain the same fields as OpenMAX IL 1.0 structures. New fields are added to the end of the structures. Thus an OpenMAX IL 1.0 structure can be interpreted as a clipped OpenMAX IL 1.1 structure if the nSize and nVersion fields are used to detect the difference.~~

~~Versions apply per method call as indicated in the nVersion field of structures passed on that call. OpenMAX IL uses the nVersion field in structures to allow the same methods to vary from version to version. Functions defined in both versions but that do not pass versioned structures are identical across versions.~~

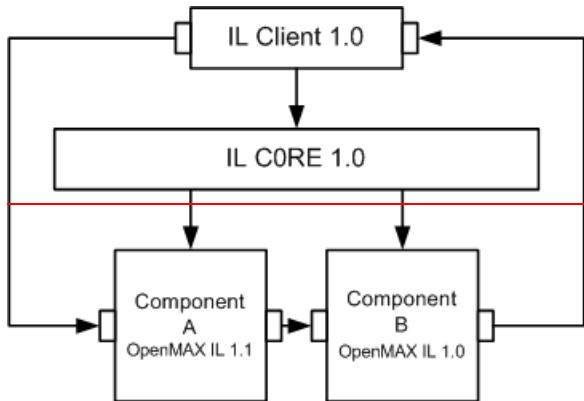
~~In addition, enumerated values remain consistent with enumerated values from OpenMAX IL 1.0.~~

~~Lastly parameters to methods defined in OpenMAX IL 1.0 remain unchanged in OpenMAX IL 1.1. Specifically the number and format of the parameter remains unchanged, but additional fields may be added to the contents of the parameters. Method functionality defined in OpenMAX IL 1.0 remains unchanged by default (i.e. a new method or parameter must be used to enable OpenMAX IL 1.1 functionality).~~

~~The following section details how heterogeneous pieces operate together. The section simplifies the combinations and permutations of heterogeneous pieces into systems with IL clients using OpenMAX IL 1.0 methods, and IL clients using OpenMAX IL 1.1 methods.~~

#### ~~IL Client 1.0~~

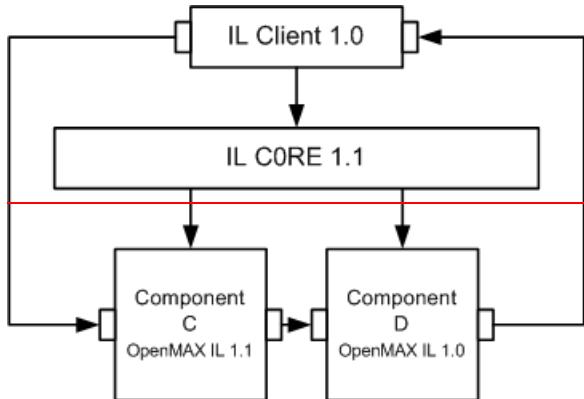
~~Backward compatibility requires IL clients using OpenMAX IL 1.0 methods to work with a core, and components from OpenMAX IL 1.1. Furthermore, an OpenMAX IL 1.1 core needs to operate with both OpenMAX IL 1.0 and OpenMAX IL 1.1 components.~~



**Figure 1-2. OpenMAX IL 1.0 Client Using 1.0 Core and 1.0 & 1.1 Components**

In the above simple example, an client using OpenMAX IL 1.0 methods is operating with an OpenMAX IL 1.0 core and components from both OpenMAX IL 1.0, and OpenMAX IL 1.1.

The OpenMAX IL 1.1 component A is the only OpenMAX IL 1.1 piece in the system. Component A detects the Core is OpenMAX IL 1.0 by the value 1.0.R.S in the nVersion field of the component handle OMX\_COMPONENTTYPE. Component A uses this version information to set OpenMAX IL 1.0 compatible interfaces for all method pointers in the component handle. Component A further uses this version information to not issue calls to any of the core methods new to OpenMAX IL 1.1 (e.g. content pipes).



**Figure 1-3. OpenMAX IL 1.0 Client Using 1.1 Core and 1.0 & 1.1 Components**

In the above complex example, a client using OpenMAX IL 1.0 methods is operating with an OpenMAX IL 1.1 core and components from both OpenMAX IL 1.0, and OpenMAX IL 1.1.

The OpenMAX IL 1.1 core, determines that Component D, is an OpenMAX IL 1.0 component via the OMX\_GetComponentVersion method. The core should flag component D as OpenMAX IL 1.0, and allocate the component handle in OMX\_COMPONENTTYPE as an OpenMAX IL 1.0 structure. All subsequent method accesses to Component D from the core shall be restricted to OpenMAX IL 1.0 methods. Furthermore the core shall provide OpenMAX IL 1.0 structures for all methods to component D.

The OpenMAX IL 1.1 component C detects the core is OpenMAX IL 1.1 by the version of the component handle in OMX\_COMPONENTTYPE. Component C may use OpenMAX IL 1.1 core methods.

Component C, detects per method that the client is using OpenMAX IL 1.0 methods and structures and responds accordingly.

Lastly Component C detects Component D is OpenMAX IL 1.0 during the ComponentTunnelRequest method to setup the tunnel between components C and D, by inspecting the nVersion field in the component handle provided in hTunneledComp. Component C then uses OpenMAX IL 1.0 methods and structures for the tunnel with Component D.

#### IL Client 1.1

Clients developed for OpenMAX IL 1.1 have visibility into this version, and prior versions of the specification. It is expected that these clients will use the version information provided by the OMX\_GetComponentVersion method to determine the version of OpenMAX IL supported by each component.

The Client may use OpenMAX IL 1.0 interfaces for OpenMAX IL 1.0 components. The client should not use OpenMAX IL 1.1 interfaces on a OpenMAX IL 1.0 component, if the client chooses to do so the behavior is not defined.

## **2 OpenMAX IL Introduction and Architecture**

This section of the document describes the OpenMAX IL features and architecture. The OpenMAX IL layer is an API that defines a software interface used to provide an access layer around software components in a system. The intent of the software interface is to take components with disparate initialization and command methodologies and provide a software layer that has a standardized command set and a standardized methodology for construction and destruction of the components.

### **2.1 Architectural Overview**

Consider a system that requires the implementation of four multimedia processing functions denoted as F1, F2, F3, and F4. Each of these functions may be from different vendors or may be developed in house but by different groups within the organization. Each may have different requirements for setup and teardown. Each may have different methods of facilitating configuration and data transfer. The OpenMAX IL API provides a means of encapsulating these functions, singly or in logical groups, into components. The API includes a standard protocol that enables compliant components that are potentially from different vendors/groups to exchange data with one another and be used interchangeably.

The OpenMAX IL API interfaces with a higher-level entity denoted as the IL client, which is typically a functional piece of a filter graph multimedia framework, OpenMAX AL, or an application. The IL client interacts with a centralized IL entity called the core. The IL client uses the OpenMAX IL core for loading and unloading components, setting up direct communication between two OpenMAX IL components, and accessing the component's methods.

An IL client always communicates with a component via the IL core. In most cases, this communication equates to calling one of the IL core's macros, which translates directly to a call on one of the component methods. Exceptions (where the IL client calls an actual core function that works) include component creation and destruction, queries about installed components and the roles they support, and connection via tunneling of two components.

Components embody the media processing function or functions. Although this specification clearly defines the functionality of the OpenMAX IL core, the component provider defines the functionality of a given component. Components operate on four types of data that are defined according to the parameter structures that they export: audio, video, image, and other (e.g., time data for synchronization).

An OpenMAX IL component provides access to a standard set of component functions via its component handle. These functions allow a client to get and set component and port configuration parameters, get and set the state of the component, send commands to the component, receive event notifications, allocate buffers, establish communications with a single component port, and establish communication between two component ports.

Every OpenMAX IL component shall have at least one port to claim OpenMAX IL conformance. Although a vendor may provide an OpenMAX IL-compatible component without ports, the bulk of conformance testing is dependent on at least one conformant port. The four types of ports defined in OpenMAX IL correspond to the types of data a port may transfer: audio, video, and image data ports, and other ports. Each port is defined as either an input or output depending on whether it consumes or produces buffers.

In a system containing four multimedia processing functions F1, F2, F3, and F4, a system implementer might provide a standard OpenMAX IL interface for each of the functions. The implementer might just as easily choose any combination of functions. The delineation for the separation of this functionality is based on ports. Figure 2-1 shows a few possible partitions for an OpenMAX IL implementation that provides these functions.

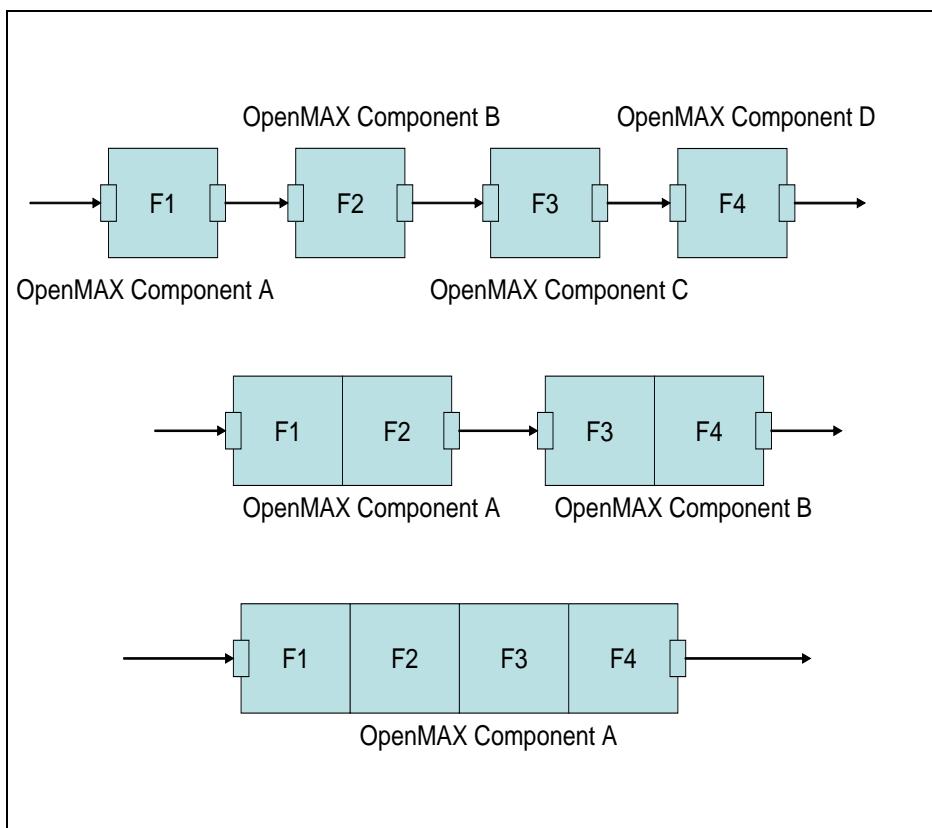


Figure 2-1. Possible Partitions for an OpenMAX IL Implementation

## 2.2 Key Vocabulary

This section describes acronyms and definitions commonly used in describing the OpenMAX IL API.

### 2.2.1 Key Definitions

Table 2-1 lists key definitions used in describing the OpenMAX IL API.

**Table 2-1: Key Definitions**

Key word	Meaning
Accelerated component	OpenMAX IL components that wrap a function with a portion running on an accelerator.
Accelerator	Hardware designed to speed up processing of some functions. This hardware may also be referred to as accelerated hardware. Note that the accelerator may actually be software running in a different processor and not be hardware at all.
<u>Allocator port</u>	<u>An allocator port is a port that allocates its own buffers.</u> <u>An allocator port is a supplier port that also allocates its own buffers.</u>
Buffer Supplier	<u>The entity that requests the buffer headers to be allocated, i.e. the entity (port or IL client) invoking either UseBuffer or AllocateBuffer calls.</u> <u>The entity that “owns” the buffer passed into a port.</u>
Container	A format for encapsulating elementary streams of data and associated metadata (e.g. the 3gp file format).
Content Pipe	The abstraction of a means to access (read or write) some content external to OpenMAX IL. Content may manifest itself as a file and a pipe may leverage system file i/o functions, but the abstraction is not limited to these particular types of content or content access.
Component Group	A group of components <u>that are</u> <u>that is</u> functionally dependent upon one another. If one component of a group is inoperable then all components in a group are inoperable.
Component Suspension	A component is suspended when it lacks a critical resource but holds all other resources so that, if and when the required resource is again available, that component may resume from the point of suspension.
Dynamic resources	Any component resources that are allocated after the initial transition to the idle state. Dynamic resource allocation is discouraged and should only occur when the parameters of the allocation (e.g. the size or number of internal memory buffers) is not known at the preferred times to allocate resources.
Host processor	The processor in a multi-core system that controls media acceleration and typically runs a high-level operating system.
IL client	The layer of software that invokes the methods of the core or component. The IL client may be a layer below the GUI application, such as GStreamer, or may be several layers below the GUI layer. In this document, the application refers to any software that invokes the OpenMAX IL methods.
Main memory	Typically external memory that the host processor and the accelerator share.

Key word	Meaning
<u>Non-supplier port</u>	<u>A port that is not a supplier port, i.e. the port requested to allocate the buffer headers by receiving either UseBuffer or AllocateBuffer calls. The port that receives the UseBuffer call from its neighbour when a pair of ports is tunneling is the non-supplier port.</u>
OpenMAX IL component	A component that is intended to wrap functionality that is required in the target system. The OpenMAX IL wrapper provides a standard interface for the function being wrapped.
OpenMAX IL core	Platform-specific code that has the functionality necessary to locate and then load an OpenMAX IL component into main memory. The core also is responsible for unloading the component from memory when the application indicates that the component is no longer needed. In general, after the OpenMAX IL core loads a component into memory, the core will not participate in communication between the application and the component.
Resource manager	A software entity that manages hardware resources in the system.
<u>Sharing port</u>	<u>A supplier port that re-uses buffers allocated by another port.' or 'A supplier port that re-uses buffers provided by another port within the same component. A port that re-uses buffers allocated by another port within the same component is a sharing port.</u>
Static resources	Component resources that are allocated as a prerequisite to entering the idle state. Most component resources fall into this category.
<u>Supplier Port</u>	<u>A port with a role of Buffer Supplier (as defined above). The port that calls UseBuffer on its neighbour when a pair of ports is tunneling is the supplier port.</u>
Synchronization	A mechanism for gating the operation of one component with another.
Tunnels/Tunneling	The establishment and use of a standard data path that is managed directly between two OpenMAX IL components.
<u>Tunnelling port</u>	<u>A tunnelling port is one of a pair of ports that has established a standard data path, or tunnel.</u>

## 2.3 System Components

Figure 2-2 depicts the various types of communication enabled with OpenMAX IL. Each component can have an arbitrary number of ports for data communication. Components with a single output port are referred to as source components. Components with a single input port are referred to as sink components. Components running entirely on the host processor are referred to as host components. Components running on a loosely coupled accelerator are referred to as accelerator components. OpenMAX IL may be integrated directly with an application or may be integrated with multimedia framework components enabling heterogeneous implementations.

Three types of communication are described. Non-tunneled communications defines a mechanism for exchanging data buffers between the IL client and a component.

Tunneling defines a standard mechanism for components to exchange data buffers directly with each other in a standard way. Proprietary communication describes a proprietary mechanism for direct data communications between two components and may be used as an alternative when a tunneling request is made, provided both components are capable of doing so.

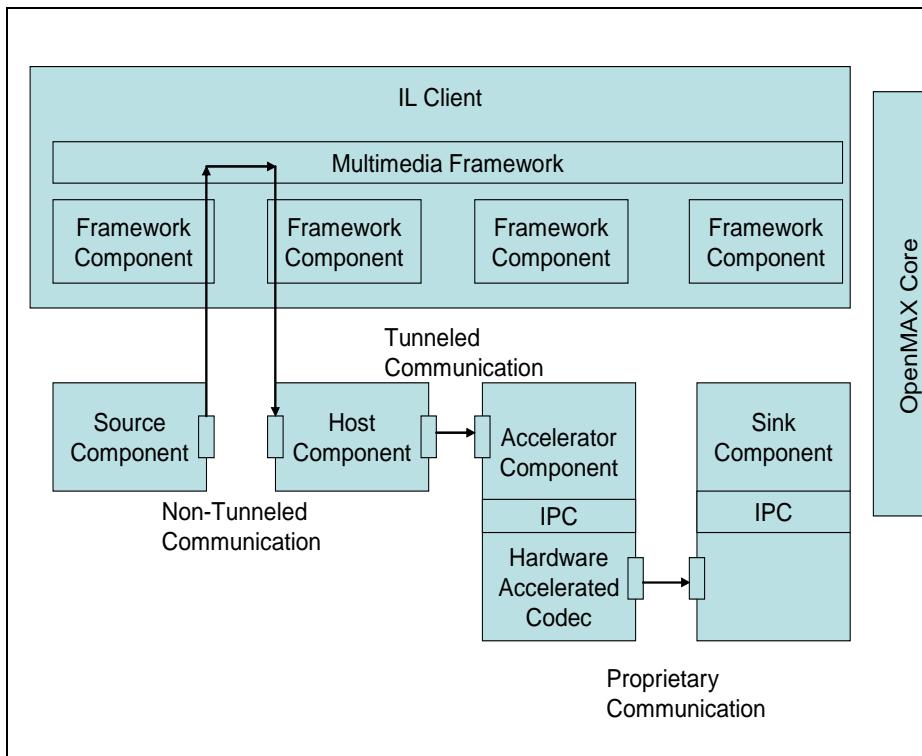


Figure 2-2. OpenMAX IL API System Components

### 2.3.1 Component Profiles

OpenMAX IL component functionality is grouped into two profiles: base profile and interop profile.

The base profile shall support non-tunneled communication. Base profile components may support proprietary communication. Base profile components do not support tunneled communication.

The interop profile is a superset of the base profile. An interop profile component shall support non-tunneled communication and tunneled communication. An interop profile component may support proprietary communication.

The primary difference between the interop profile and the base profile is that the component supports tunneled communication. The base profile exists to reduce the

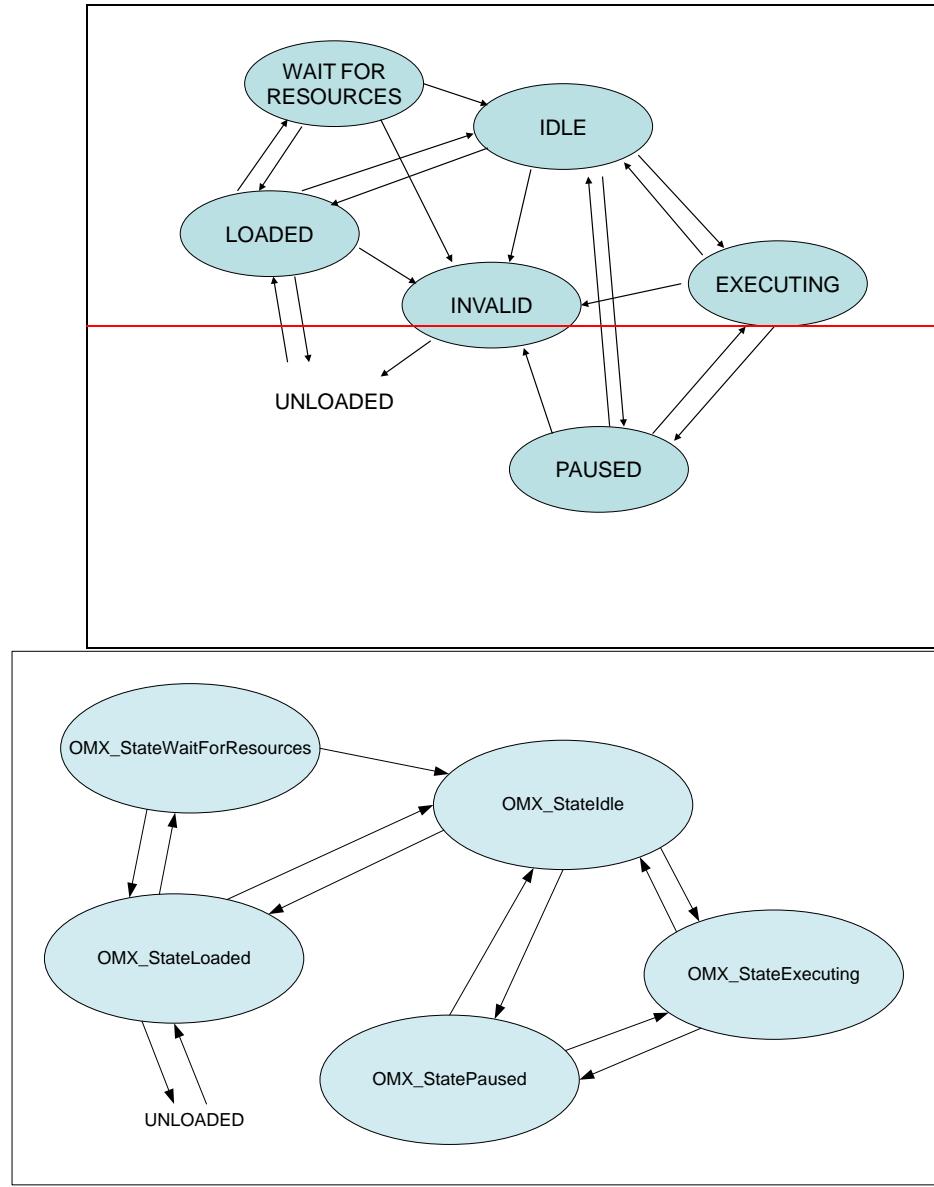
adoption barrier for OpenMAX IL implementers by simplifying the implementation. [A base profile component does not need to implement tunneled communication.](#)

**Table 2-2: Types of Communication Supported Per Component Profile**

Type of Communication	Base Profile Support	Interop Profile Support
Non-Tunneled Communication	Yes	Yes
Tunneled Communication	No	Yes
Proprietary Communication	Yes	Yes

## 2.4 Component States

Each OpenMAX IL component can undergo a series of state transitions, as depicted in Figure 2-3. Every component is first considered to be unloaded. The component shall be loaded through a call to the OpenMAX IL core. All other state transitions may then be achieved by communicating directly with the component.



**Figure 2-3. Component States**

In general, the component should all have all its operational resources when in the OMX\_Statedle, OMX\_StatePause or OMX\_StateExecuting states. There are, however, exceptions when the parameters for the resource allocation are not known at the time of the transition to OMX\_Statedle. For example, a component that decodes

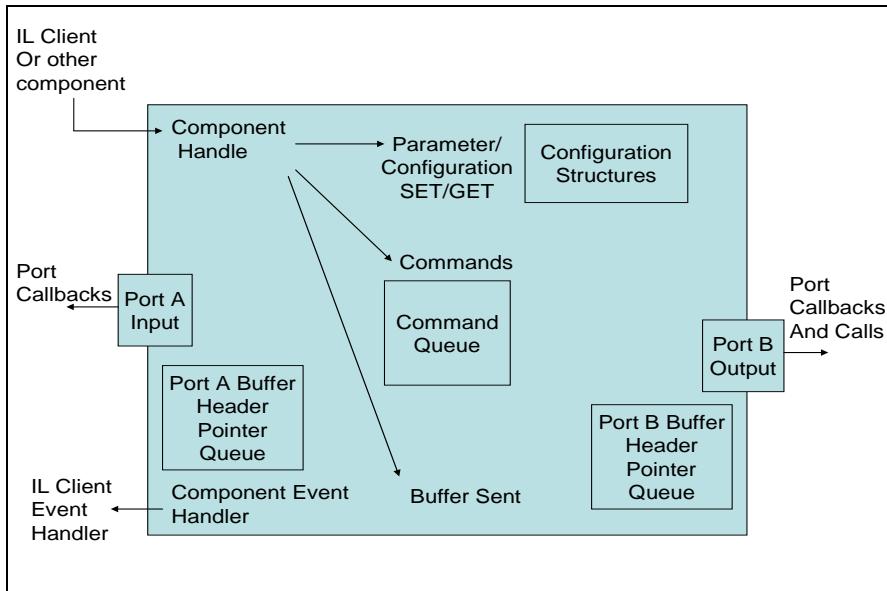
video does not know how many reference frames are required until the data stream is examined yet the component cannot examine the stream prior to transition to OMX\_StateIdle. In these cases the component may defer the allocation of resources until such time as it knows the parameters of allocation. If dynamic allocation fails the component may shall suspend itself, as described in Section 3.1.3.6. Thus we often distinguish between those resources allocated “up front” (e.g. on a transition to OMX\_StateIdle) and those allocated later by calling the former static resources and the latter dynamic resources.

Transitioning into the OMX\_StateIdle state may fail since this state requires allocation of all operational static resources. When the transition from OMX\_StateLoaded to OMX\_StateIdle fails, the IL client may try again or may choose to put the component into OMX\_StateWaitForResources. Upon entering OMX\_StateWaitForResources, the component registers with a vendor-specific resource manager to alert it when resources have become available. The component will subsequently transition into the OMX\_StateIdle state. A command that the IL client sends controls all other state transitions except to OMX\_StateInvalid.

The OMX\_StateIdle state indicates that the component has all of its needed static resources but is not processing data. The OMX\_StateExecuting state indicates that the component is processing data when possible pending reception of buffers to process data and will make required callbacks as specified in section 3. The OMX\_StatePause state maintains a context of buffer execution with the component without processing data or exchanging buffers. Transitioning from OMX\_StatePause to OMX\_StateExecuting enables buffer processing to resume where the component left off. Transitioning from OMX\_StateExecuting or OMX\_StatePause to OMX\_StateIdle will cause the context in which buffers were processed to be lost, which requires the start of a stream to be reintroduced. Transitioning from OMX\_StateIdle to OMX\_StateLoaded will cause operational resources such as communication buffers to be lost.

## 2.5 Component Architecture

Figure 2-4 depicts the component architecture. Note that there is only one entry point for the component (through its handle to an array of standard functions) but there are multiple possible outgoing calls that depend on how many ports the component has. Each component will make calls to a specified IL client event handler. Each port will also make calls (or callbacks) to a specified external function. A queue for pointers to buffer headers is also associated with each port. These buffer headers point to the actual buffers. The command function also has a queue for commands. All parameter or configuration calls are performed on a particular index and include a structure associated with that parameter or configuration, as depicted in Figure 2-4.

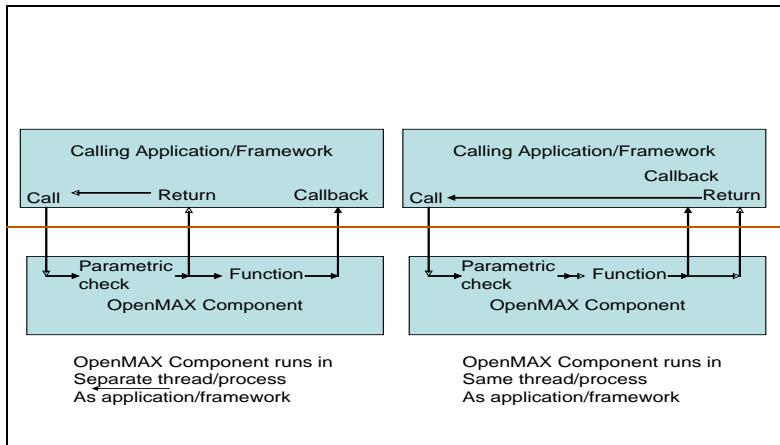


**Figure 2-4. OpenMAX IL API Component Architecture**

A port shall support callbacks to the IL client and, when part of an interop profile component, shall support communication with ports on other components.

## 2.6 Communication Behavior

Configuration of a component may be accomplished once the handle to the component has been received from the OpenMAX IL core. Data communication calls with a component are non-blocking and are enabled once the number of ports has been configured, each port has been configured for a specific data format, and the component has been put in the appropriate state. Data communication is specific to a port of the component. Input ports [receive buffers via theare always called from the IL client with OMX\\_EmptyThisBuffer call](#) (for more information, see section 3.2.2.17). Output ports [receive buffers via theare always called from the IL client with OMX\\_FillThisBuffer call](#) (for more information, see section 3.2.2.18). ~~In an in-context implementation, callbacks to EmptyBufferDone or FillBufferDone will be made before the return. Figure 2-5 depicts the anticipated behavior for an in-context versus an out-of context implementation. Note that the IL client should not make assumptions about return/callback sequences to enable heterogeneous integration of in-context and out-of context OpenMAX IL components.~~



**Figure 2-5. Out-of-Context versus In-Context Operation**

Data communications with components is always directed to a specific component port. Each port has a component-defined minimum number of buffers it shall allocate or use. A port associates a buffer header with each buffer. A buffer header references data in the buffer and provides metadata associated with the contents of the buffer. Every component port shall be capable of allocating its own buffers or using pre-allocated buffers; one of these choices will usually be more efficient than the other.

Buffers are being consumed by the component when it the component is in OMX\_StateExecuting, unless the corresponding port is disabled. A component may return a partially consumed or unconsumed buffers to the IL client or tunneled component only if:

- The IL client commands a transition from OMX\_StateExecuting or OMX\_StatePause to OMX\_StateIdle.
- Corresponding port is flushed or disabled

## 2.7 Thread Safety

OpenMAX IL cCore and Component implementations shall be thread-safe. Any of the IL functions may be called from any thread, or multiple threads simultaneously, running on either single or multi-core processors. The only restriction is being that the IL client shall not call IL cCore or cComponent functions from within an IL callback context. The IL implementations shall/must pose no special thread safety or synchronizing requirements to the calling thread, such as expecting the client to be using or setting up a specific thread synchronization mechanism.

IL callbacks may originate from any thread or multiple threads in the process that created the IL component. IL implementations shall not call client-provided callback functions from a non-thread context, such as an interrupt service routine (ISR). There are no

guarantees on how many threads the IL component is running. The IL client shall accept callbacks from any thread context, including its own. Multiple callbacks can be in progress at the same time, including callbacks to the same function. Therefore, the IL client callback implementations shall also be thread-safe.

### **2.72.8 Tunneled Buffer Allocation**

A component that is tunneling shall:

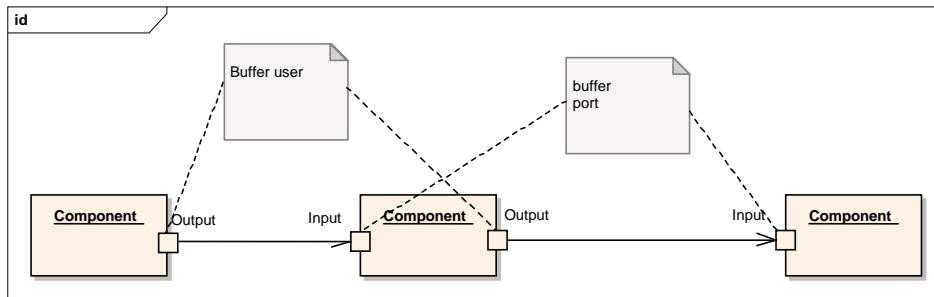
- provide buffers on all of its supplier ports;
- accurately communicate buffer requirements on all its ports;
- pass a buffer from an output port to an input port with an OMX\_EmptyThisBuffer call;
- pass a buffer from an input port to an output port with an OMX\_FillThisBuffer call.

For any given tunnel, exactly one port supplies the buffers and passes those buffers to the non-supplier port.

A port that is not tunneled and receives the AllocateBuffer call from the IL client is also considered a non-supplier port.

The set of buffer requirements for a tunneling port includes the number of buffers required, as well as the required size and alignment of each buffer.

Figure 2-5 illustrates the concepts relevant to tunneled buffer allocation.



**Figure 2-5. Port Relationships**

At any time after `OMX_SetupTunnel` completes successfully but before allocating its buffers the supplier port retrieves the tunnel's buffer requirements from its tunneled port in an `OMX_PARAM_PORTDEFINITIONTYPE` structure via an `OMX_GetParameter` call on the tunneled port's component.

It then computes the minimum buffering requirements applicable to the tunnel from its own minimum buffering requirements and those of its tunneled port.

The minimum of buffer requirements for a tunnel is the set defined as follows:

- The largest number of buffers mandated in any set. i.e. The maximum value of `nBufferCountMin` from either port
- The largest size mandated in any set. The maximum value of `nBufferSize` from either port
- The largest alignment mandated in any set. The maximum value of `nBufferAlignment` from either port

Note that, whilst normally the supplier port of a tunnel also allocates its own buffers, under the right circumstances, a **tunneling** component may choose to re-use buffers from one port on another upon the same component to avoid memory copies and optimize memory usage. This optional practice, known as buffer sharing is described in detail in Section 10—**Implementing Buffer Sharing**.

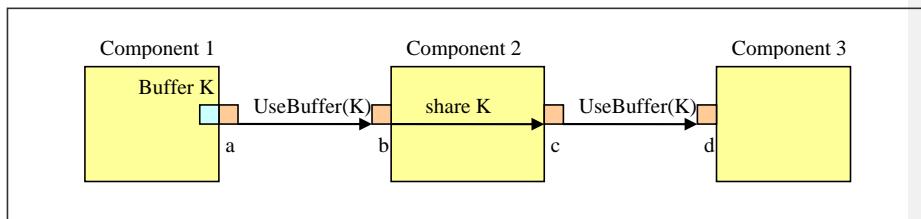
Buffer sharing may require that a tunneled port needs to include its shared port's buffer requirements in the minimum requirements calculation above determine the correct tunnel buffer requirements. However it may determine the buffer requirements from a port that shares its buffers without resorting to an OMX\_GetParameter call since they are both contained in the same component.

Section 3.4.1.2 describes tunneling setup and initialization in further detail.

This section describes buffer allocation for tunneling components. For a given tunnel, exactly one port supplies the buffers and passes those buffers to the non-supplier port. Normally the supplier port of a tunnel also allocates the buffers. Under the right circumstances, however, a tunneling component may choose to re-use buffers from one port on another to avoid memory copies and optimize memory usage. This optional practice, known as buffer sharing is described in detail in Section 11—Implementing Buffer SharingImplementing Buffer SharingImplementing Buffer Sharing.

Formatted: Indent: Left: 0.75"

Figure 2-6 illustrates the concepts relevant to tunneled buffer allocation.



**Figure 2-6. Example of Buffer Allocation and Sharing Relationships**

Among a pair of ports that are tunneling, the port that calls UseBuffer on its neighbor is known as a supplier port. A buffer supplier port does not necessarily allocate its buffers; it may re-use buffer from another port on the same component. Ports a and c in Figure 2-6 illustrate supplier ports.

The port that receives the UseBuffer calls from its neighbor is known as a non-supplier port. Ports b and d in Figure 2-6 illustrate non-supplier ports.

A port's tunneling port is the port neighboring it with which it shares a tunnel. For example, port b in Figure 2-6 is the tunneling port to port a. Likewise, port a is the tunneling port to port b.

An allocator port is a supplier port that also allocates its own buffers. Port a in Figure 2-6 is the only allocator port.

A sharing port is a port that re-uses buffers from another port on the same component. For example, port c in Figure 2-6 is a sharing port.

A tunneling component is a component that uses at least one tunnel.

The set of buffer requirements for a port includes the number of buffers required and the required size of each buffer. The maximum of multiple sets of buffer requirements is defined as the largest number of buffers mandated in any set combined with the

largest size mandated in any set. The set of buffer requirements for a port includes the number of buffers required, as well as the required size and alignment of each buffer. The maximum of multiple sets of buffer requirements is the set defined as follows:

The largest number of buffers mandated in any set

The largest size mandated in any set

The largest alignment mandated in any set

← Formatted: Indent: Left: 0.75"

One port retrieves buffer requirements from its tunneled port in a `OMX_PARAM_PORTDEFINITIONTYPE` structure via an `OMX_GetParameter` call on the tunneled port's component. Note that one port may determine buffer requirements from a port that shares its buffers without resorting to an `OMX_GetParameter` call since they are both contained in the same component.

Regardless of whether the component is sharing buffers or not, it shall is obligated to obey the following external semantics:

provide buffers on all of its supplier ports;

accurately communicate buffer requirements on its ports;

pass a buffer from an output port to an input port with an `OMX_EmptyThisBuffer` call;

Return pass a buffer from an input port to an output port with an `OMX_FillThisBuffer` call.

#### **2.7.12.8.1 IL Client Component Setup**

To set up tunneling components, the IL client should perform the following setup operations in this order:

1. Load all tunneling components and set up the tunnels on these components.
2. Command all tunneling components to transition from `OMX_StateLoaded` to `OMX_StateIdle`.

Note that if an IL client does not operate in this manner when some components are sharing buffers, a tunneling component might never transition to `OMX_StateIdle` because of the possible dependencies between components.

#### **2.7.22.8.2 Component Transition from OMX\_StateLoaded to OMX\_StateIdle**

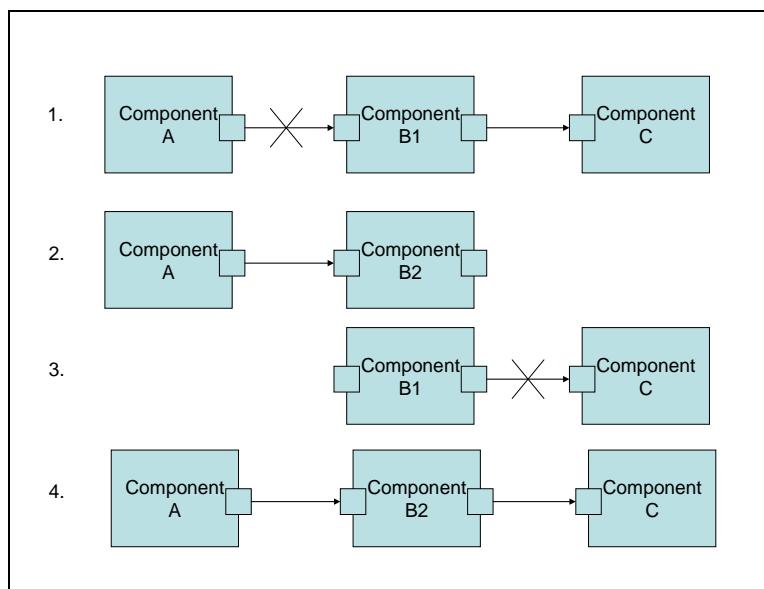
When commanded to transition from `OMX_StateLoaded` to `OMX_StateIdle`, each allocatorsupplier port of a non sharing component shall does the following:

1. Determine the buffer requirements of its tunneled port via an `OMX_GetParameter` call.

2. Allocate buffers according to the maximum of its own requirements and the requirements of the tunneled port.
- 2.3. Communicate the actual number of buffers that shall be used for the tunnel and the contiguity of those buffers via a call to `OMX_SetParameter(OMX_IndexParamPortDefinition)`, if the `nBufferCountActual` or `bBuffersContiguous` fields are different from that advertised by the non-supplier port.
- 3.4. Call `OMX_UseBuffer` on its tunneling port after it receives the `OMX_SetConfig(OMX_IndexConfigTunneledPortStatus)` call as specified in section 3.1.3.13.

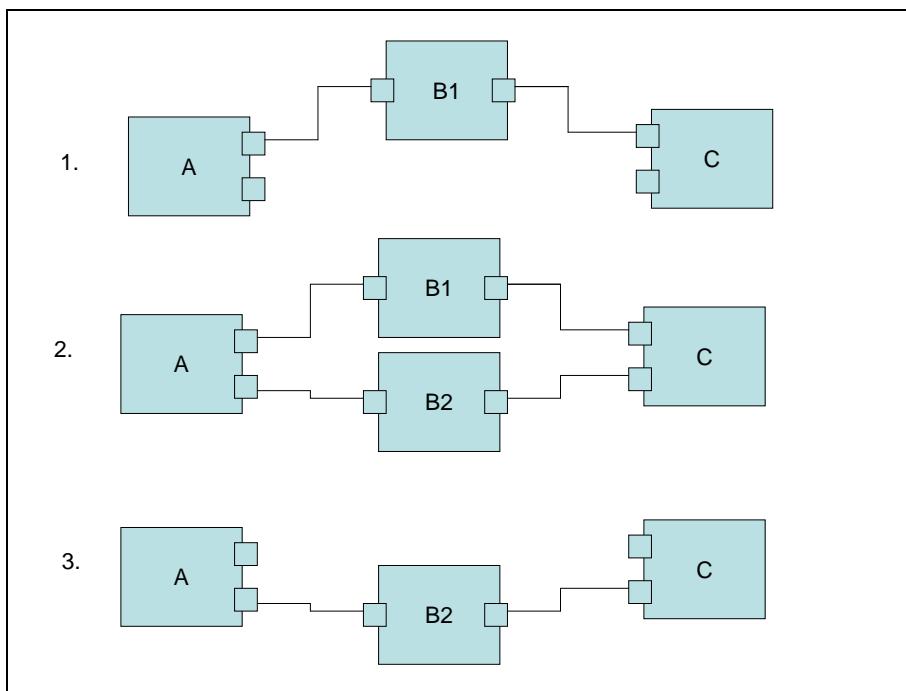
### 2.82.9 Port Reconnection

Port reconnection enables a tunneled component to be replaced with another tunneled component without having to tear down surrounding components. In [Figure 2-6](#), component B1 is to be replaced with component B2. To do this, the component A output port and the component B input port shall first be disabled with the port disable command. Once all allocated buffers have returned to their rightful owner and freed, [the tunnel between A and B1 can be torn down, and](#) the component A output port may be connected to component B2. The component B1 output port and the component C input port should similarly be given the port disable command. After all allocated buffers have returned to their owners and freed, [the tunnel between B1 and C can be torn down, and](#) the component C input port may be connected to the component B2 output port. Then all ports may be given the enable command. Refer to [Section 3.4.4 Port Disablement and Enablement](#) for additional information regarding port disabling and enabling.



**Figure 2-6. Port Recommendation**

In some cases such as audio, reconnecting one component to another and then fading in data for one component while fading out data for the original component may be desirable. [Figure 2-7](#) illustrates how this would work. In step 1, component A sends data to component B1, which then sends the data on to component C. Components A and C both have an extra port that is disabled. In step 2, the IL client first establishes a tunnel between component A and B2, then establishes a tunnel between B2 and C, and then enables all ports in the two tunnels. Component C may be able to mix data from components B1 and B2 at various gains, assuming that these are audio components. In step 3, the ports connected to component B1 from components A and C are disabled, [the tunnels involving B1 are torn down](#), and component B1 resources may be de-allocated.



**Figure 2-7. Reconnecting Components**

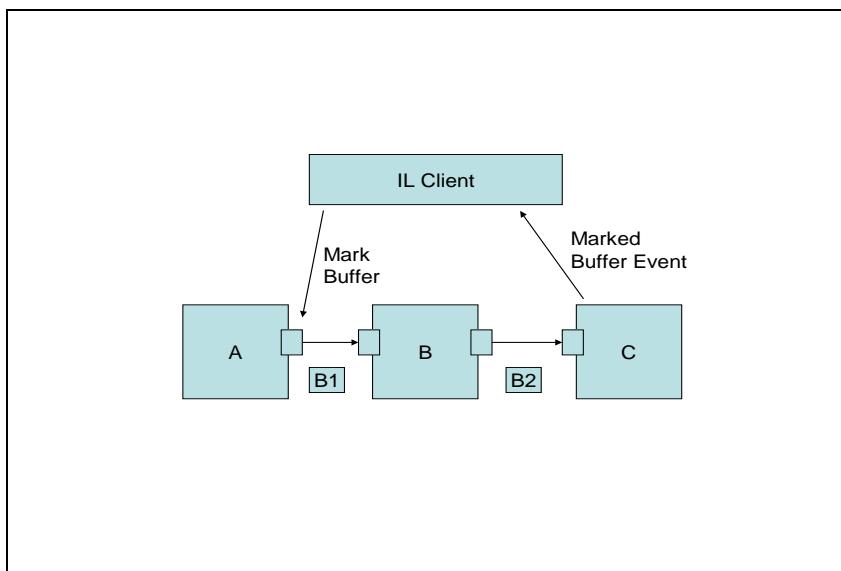
### **2.92.10 Queues and Flush**

A [flush command causes separate command queue enables](#) the component to flush buffers that have not been processed and return these buffers to the IL client when using non-tunneled communication, or to the [supplier tunneled output](#) port when using tunneled communication. For example, [assume](#) that a component has an output port that is using buffers allocated by the IL client. In this example, the client sends a series of five buffers to the component before sending the flush command. Upon processing the flush

command, the component returns each unprocessed buffer and triggers its event handler to notify the IL client. Two buffers were already processed before the flush command got processed. The component returns the remaining three buffers unfilled and generates an event. The IL client should wait for the event before attempting to de-initialize the component.

### **2.102.11 Marking Buffers**

An IL client can also trigger an event to be generated when a marked buffer is encountered. A buffer can be marked in its buffer header. The mark is internally transmitted from an input buffer to an output buffer in a chain of OpenMAX IL components. The mark enables a component to send an event to the IL client when the marked buffer is encountered. [Figure 2-8](#) depicts how this works.



**Figure 2-8. Marking Buffers**

The IL client sends a command to mark a buffer [from the output port of Component A, with Component C as the target](#). The next buffer, [B1](#), sent from [this port](#) [the output port of the component](#) is marked [B1](#). Component B processes [the B1-buffer](#) and provides the results in buffer [B2](#), along with the mark. [After](#) [When](#) component C [has processed](#) [receives the marked buffer](#) [B2](#) [through its input port](#), the component does not trigger its event handler [with the buffer mark](#), until it has processed the buffer.

### **2.112.12 Events and Callbacks**

[Several](#) kinds of events are sent by a component to the IL client:

- *Error events* are enumerated and can occur at any time.

- *Command complete notification events* are triggered upon successful or unsuccessful execution of a command.
- *Marked buffer events* are triggered upon detection of a marked buffer by a component.
- *Settings changed notification events* are generated when the component changes its port-settings.
- A *buffer flag event* is triggered when an end of stream is encountered.
- *RAresources acquired events* are generated when a component has a change in its resource status, gets resources that it has been waiting for.
- *Port command events* are generated when further IL client action is needed to complete an ongoing command.

Ports make buffer handling callbacks upon availability of a buffer or to indicate that a buffer is needed.

### 2.122.13 Buffer Payload

The port configuration is used to determine and define the format of the data to be transferred on a component port, but the configuration does not define how that data exists in the buffer.

There are generally three cases that describe how a buffer can be filled with data. Each case presents its own benefits.

In all cases, the range and location of valid data in a buffer is defined by the pBuffer, nOffset, and nFilledLen parameters of the buffer header. The pBuffer parameter points to the start of the buffer. The nOffset parameter is set by the entity that places data into the buffer (or forwards data in a shared buffer) to indicate the number of bytes between the start of the buffer and the start of valid data. The nFilledLen parameter specifies the number of contiguous bytes of valid data in the buffer. The valid payload data in the buffer is therefore located in the range pBuffer + nOffset to pBuffer + nOffset + nFilledLen. Metadata may be present in the buffer after this region.

The following cases are representative of compressed data in a buffer that is transferred into or out of a component when decoding or encoding. In all cases, the buffer just provides a transport mechanism for the data with no particular requirement on the content. The requirement for the content is defined by the port configuration parameters.

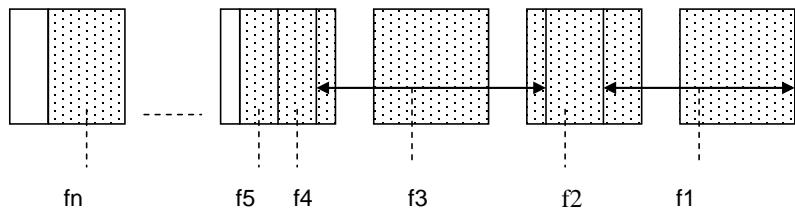
The shaded portion of the buffer represents data and the white portion denotes no data.

Case 1: Each buffer is filled in whole or in part. In the case of buffers containing compressed data frames, the frames are denoted by f1 to fn.

#### Case 1

buffer n	buffer 4	buffer 3	buffer 2	buffer 1
----------	----------	----------	----------	----------

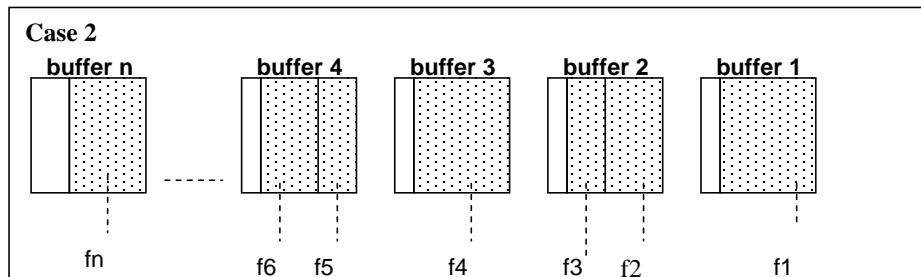




**Figure 2-9: Case 1—Each Buffer Filled In Whole or In Part**

Case 1 provides a benefit when decoding for playback. The buffer can accommodate multiple frames and reduce the number of transactions required to buffer an amount of data for decoding. However, this case may require the decoder to parse the data when decoding the frames. It also may require the decoder component to have a frame-building buffer in which to put the parsed data or maintain partial frames that would be completed with the next buffer.

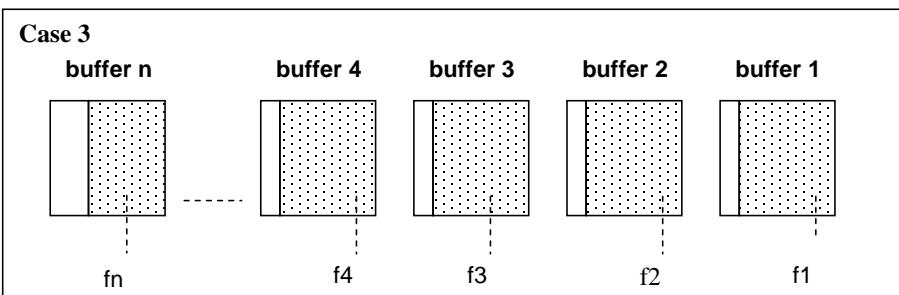
Case 2: Each buffer is filled with only complete frames of compressed data.



**Figure 2-10: Case 2—Each Buffer Filled with Only Complete Frames of Data**

Case 2 differs from case 1 because it requires the compressed data to be parsed first so that only complete frames are put in the buffers. Case 2 may also require the decoder component to parse the data for decoding. This case may not require the extra working buffer for parsing frames required in case 1.

Case 3: Each buffer is filled with only one frame of compressed data.



**Figure 2-11: Case 3—Each Buffer Filled with Only One Frame of Compressed Data**

The benefit in case 3 is that a decoding component does not have to parse the data. Parsing would be required at the source component. However, this method creates a bottleneck in data transfer. Data transfer would be limited to one frame per transfer. Depending on the implementation, one transaction per frame could have a greater impact on performance than parsing frames from a buffer.

At a minimum, a decoder or encoder component would be required to support case 1. By definition, if a codec component can support case 1, then it can support cases 2 and 3, but only if the compression format allows for byte-aligned frame boundaries. Operating in case 2 or 3 may not make sense when, for example, configuring an Adaptive Multi-Rate (AMR) codec for RTP-payload format, bandwidth-efficient mode. The non-byte aligned frames defined by this format would not fit the byte-aligned frame boundaries defined by these cases.

When filling a buffer with compressed data for input to a decoder or output from an encoder, a problem with limiting the filling to complete frames only might arise when

frames are not byte aligned. Padding would have to be added outside of any padding defined in the format specification. The padding would then need to be removed, since the data could not be appended as is. This would require knowledge of the padding bits outside of any standard specification. Likewise, if this padding were not in place to maintain compliance with the standards specification for the port configuration, complete frames could not always be placed in the buffers. In either case, specific knowledge of how this situation is handled would be required, and may be different between components.

For interoperability, the content delivered in a buffer should not be assumed or required to be any number of complete frames, although at least one complete unit of data will be delivered in a buffer for uncompressed data formats. Compressed data formats do not place restrictions on the amount of content delivered in each buffer.

There are a number of codecs that do not include frame boundary signaling in the elementary bitstream, instead relying on container signaling. Examples of this are WMV Simple and Main Profiles, VP8 and Real Video. For such cases it is expected that the component/source entity, up stream of the decoder (like demuxer/parser) supplies the data with frame boundaries signaled (for example through use of the OMX\_BUFFERFLAG\_ENDOFFRAME and OMX\_BUFFERFLAG\_SKIPFRAME flag).

## **2.14 Signalling frames and subframes**

To aid processing of data the producer of data can explicitly signal frames or subframes that appear in a buffer payload. These flags can be applied to a buffer, and signal that the end of the valid payload in that buffer ends a frame or a subframe.

### **2.14.1 Signalling frames**

**Benefits of Using OMX\_BUFFERFLAG\_ENDOFFRAME**

**Provides a mechanism for decoder components to identify frame boundaries when bit-stream contains no sync-words or start-codes**

In the use-cases where frames are fragmented over multiple buffer payloads, as is often the case in streaming, the usage of the optional OMX\_BUFFERFLAG\_ENDOFFRAME flag (defined in 3.1.3.7.1, Parameter Definitions) provides a mechanism to easily identify the end of a frame boundary and the beginning of the next frame without the need for additional processing and bit-stream parsing. This is especially useful if the frames are fragmented over multiple buffer payloads and bit-stream format does not contain sync-words, start-codes or other means of identifying frame boundaries (e.g. WMV simple/main profile frames fragmented over multiple buffer payloads).

**Eliminates the need to wait for the next buffer to detect the end of frame**

Even if the bit-stream contains other means for identifying frame boundaries (e.g. through presence of start-codes or sync-words), the decoder component must still wait for

the arrival of the next buffer to clearly identify the end of the previous frame and the beginning of the next frame.

The usage of the optional OMX\_BUFFERFLAG\_ENDOFFRAME flag eliminates the need for waiting for the next buffer and allows the decoder component to start decoding the bit-stream immediately after receiving the buffer marked with OMX\_BUFFERFLAG\_ENDOFFRAME flag.

Referring to Figure 2-12 as an example, with the presence of OMX\_BUFFERFLAG\_ENDOFFRAME flag, the decoder component can detect the end of Frame 1 immediately after receiving Buffer 3. On the other hand, in the absence of OMX\_BUFFERFLAG\_ENDOFFRAME flags, the decoder component would have to wait to receive at least Buffer 4 as well before being able to detect the end of Frame 1 (even if start-codes or sync-words are present in the bitstream).

#### **2.14.2 Signalling subframes****Benefits of Using OMX\_BUFFERFLAG\_ENDOFSUBFRAME**

Several video compression formats define usage of sub-frames (independently decodable units smaller than a frame, or decodable units that represent side information). For example, in the ITU H.264\AVC specification, video content is divided into NALUs, a set of NALUs representing a coded picture or other side information. Similarly, in the case of MPEG4 [MPEG-4 Visual v2] or H263 bit-streams [H.263, RFC4629] – a decoder may use a picture slice / GOB as an independently decodable unit. In the case of VC-1 Advanced Profile [VC1], a picture slice is independently decodable, as well as a field in case of interlaced content of VC-1 Advanced Profile.

The benefits of using OMX\_BUFFERFLAG\_ENDOFSUBFRAME (defined in 3.1.3.7.1) are in some aspects similar to the benefits of using OMX\_BUFFERFLAG\_ENDOFFRAME, except that they apply to sub-frames rather than frames. In some aspects, however, the OMX\_BUFFERFLAG\_ENDOFSUBFRAME flag has its own unique benefits.

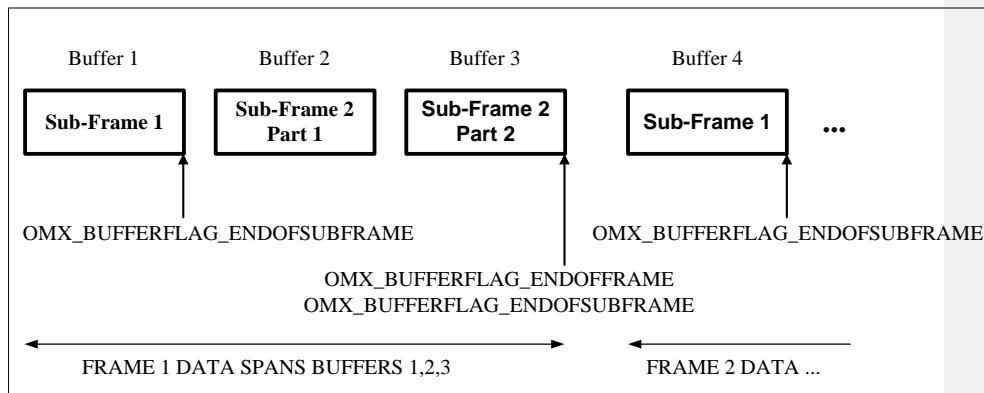
##### **2.14.2.1 Provides a mechanism for decoder components to identify sub-frame boundaries when bit-stream contains no sync-words or start-codes**

Firstly, iIn the case where NALUs are provided without Start Codes, as is the case of RTP streaming of H264 [Section 4.3.32] (i.e. when NALU format is “OMX\_NaluFormatOneNaluPerBuffer”),

OMX\_BUFFERFLAG\_ENDOFSUBFRAME flag provides a mechanism to identify the end of a NALU and the beginning of the next NALU, specifically for the case where a NALU is fragmented over multiple RTP payloads i.e. buffers. Without this mechanism, it is impossible for the decoder component to identify the boundaries of the NALU, since the bit-stream itself contains no sync-words, start-codes or other clues that would allow the decoder component to reconstruct the NALU boundary.

### **Eliminates the need to buffer the entire frame before processing**

Secondly, If the output port of the source component implements and sets the `OMX_BUFFERFLAG_ENDOFSUBFRAME` flag, the decoder components that support decoding of sub-frames can start decoding the bit-stream immediately after receiving a buffer marked with `OMX_BUFFERFLAG_ENDOFSUBFRAME` flag, rather than waiting for the entire frame to buffer up. Waiting to receive the entire frame may take several buffer payloads and may incur a delay penalty. Referring to Figure 2-12 as an example, the decoder component that supports sub-frame decoding can start decoding immediately after receiving Buffer 1, rather than buffering Buffers 1 through 3 (which all belong to Frame 1).



**Figure 2-12: Example usage of `OMX_BUFFERFLAG_ENDOFFRAME` and `OMX_BUFFERFLAG_ENDOFSUBFRAME` flags**

### **Eliminates the need to wait for the next buffer to detect the end of subframe**

Finally, Even if the bit-stream contains other means for identifying sub-frame boundaries (e.g. through presence of start codes), the decoder component must still wait for the arrival of the next buffer to clearly identify the end of the previous sub-frame and the beginning of the next sub-frame.

The usage of the optional `OMX_BUFFERFLAG_ENDOFSUBFRAME` flag eliminates the need for waiting for the next buffer and allows the decoder component that supports sub-frame decoding to start decoding the bit-stream immediately after receiving the buffer marked with `OMX_BUFFERFLAG_ENDOFSUBFRAME` flag.

Again, referring to Figure 2-12 as an example, with the presence of `OMX_BUFFERFLAG_ENDOFSUBFRAME`, the decoder component can detect the end of Sub-frame 1 immediately after receiving Buffer 1. On the other hand, in the absence of `OMX_BUFFERFLAG_ENDOFSUBFRAME` flags, the decoder component would have to wait to receive at least Buffer 1 and Buffer 2 before being able to detect the end of Sub-frame 1, even if start-codes are present in the bit-stream.

This benefit is similar to the benefit of using the `OMX_BUFFERFLAG_ENDOFFRAME` flag, except that it applies to sub-frames, rather than frames.

### **2.132.15 Buffer Flags and Timestamps**

Buffer flags associate certain properties (e.g., the end of a data stream) with the data contained in a buffer. A buffer timestamp associates a presentation time in microseconds with the data in the buffer used to time the rendering of that data. Once a timestamp is associated with a buffer, no component should alter the timestamp for rate control or synchronization, which are implemented in the clock component.

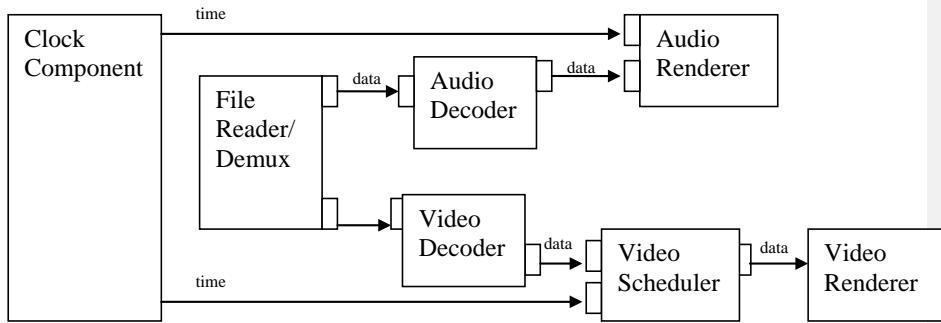
Buffer metadata (i.e., flags and timestamps) applies to the first new logical unit in the buffer. Thus, given the presence of multiple logical units in a buffer, the metadata applies to the logical unit whose starting boundary occurs first in the buffer. Subsequent logical units in a buffer don't have explicit flags or timestamps. If explicit flag and timestamps are required on every logical unit, at most one one or less-logical unit should be included in each buffer. Unless otherwise stated (e.g., in a flag definition), a component that receives a logical input unit marked with a flag or timestamp shall copy that metadata to all logical output units that the input contributes to.

### **2.142.16 Synchronization**

Synchronization is enabled by the use of synchronization (sync) ports on a clock component. These ports and the clock component are defined within the “other” domain and operate with the same protocols and calls that regulate data ports. The clock component maintains a media clock that tracks the position in the media stream based on audio and video reference clocks. The clock component calls `OMX_SetConfig` or transmits buffers containing time information (denoted by a media time update and containing the media clock's current position, scale, and state) to client components via sync ports. A client component may time the execution of an operation (e.g., the presentation of a video frame) to a timestamp by requesting that the clock component send that timestamp when it matches the media clock. In this case, the client component executes the operation when it receives the fulfillment of the request over its sync port.

Formatted: Body Text

Figure 2-13 illustrates the flow of time and information in data buffers in an example configuration of components.



**Figure 2-13. Flow of Time and Data Buffers**

### **2.152.17 Rate Control**

The clock component also implements all rate control by exposing a set of configurations for controlling its media clock. The IL client may change the scale factor of the media clock (effectively changing the rate and direction that the media clock advances) to implement play, fast forward, rewind, pause, and slow motion trick modes. The IL client may also start and stop the clock by using these configurations to change the state of the media clock. The clock component makes all of its client components aware of a change to the media clock scale and state by sending a media time update with the new scale or state on all sync ports. Although a component may not alter a buffer timestamp in reaction to a scale change, a component may alter its processing accordingly. For instance, an audio component might scale and pitch correct audio during trick modes or cease transmitting output entirely.

### **2.162.18 Component Registration**

How components are registered with a core is generally core specific.

However, if the core supports static linking with components, then it will support a standard compile-time component registration scheme as described in Section 3.1.3.1. Vendors can therefore supply components that are suitable for static linking with all cores that support it; this is achieved by placing component information into a data structure that is linked with the component and the core.

A component can be registered statically using this mechanism but have the bulk of its code dynamically loaded.

A component supplies an interface for retrieving the standard component roles it supports. The core may leverage this interface for exposing role-related information to the IL client.

### **2.172.19 Resource Management**

This section discusses the role of resource management in the OpenMAX IL API.

### **2.17.12.19.1 Need for Resource Management**

When a component ~~fails~~ is not allowed to go to OMX\_StateIdle due to lack of resources, the IL client ~~may~~ does not know what the limited resource is or which components are using that resource. Therefore, the IL client cannot resolve the resource conflict. These situations necessitate IL resource management.

One of the goals of OpenMAX IL is hardware independence provided by the IL layer to the layers above it. The goal of hardware independence can be achieved by specifying the following requirements regarding resource management:

- An IL client (e.g., a multimedia plug-in that is typically part of a software platform) should not need to know the details of an IL implementation or which resource an IL component is using.
- In case of resource conflicts, an IL client should be able to rely on consistent component behavior across IL implementations and hardware platforms.
- An IL client should not have to interface directly with a hardware vendor-specific resource manager for two reasons.
  - This method violates the goal of hardware independence.
  - This method adds considerable re-work to the IL client, which has an impact on the re-usability of the IL client on multiple hardware platforms.

Although resource management is not fully addressed in OpenMAX IL API version 1.24, “hooks” for resource management ~~have~~ ~~has~~ been put in place in the form of behavioral rules, component priorities, and a resource management-related component state. These “hooks” lay the groundwork for full-fledged resource management in later versions of the OpenMAX IL API.

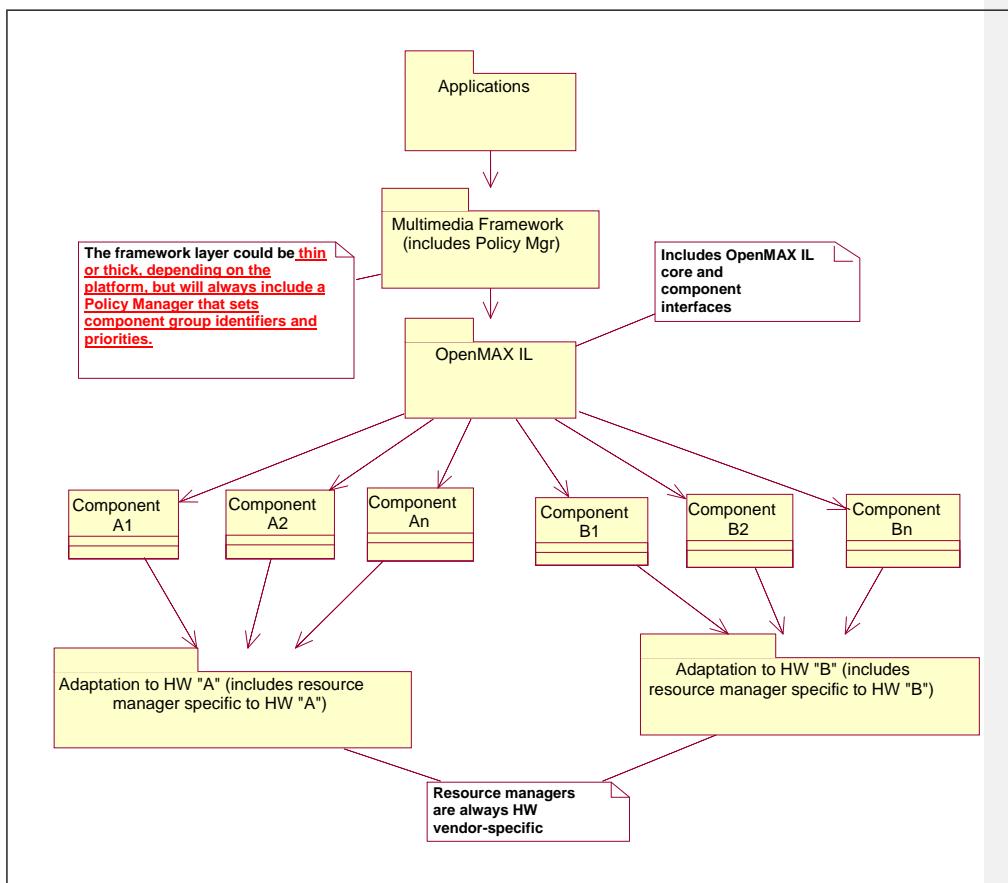
Before proceeding further, the terms resource management and policy are defined for the benefit of the discussion that follows:

- *A Resource manager* is responsible for ~~controlling the access of components to a resource. It is aware of how much of each resource any component is using, and how much is available. It will grant, and pre-empt, access to resources based on availability, and the component priorities managing the access of components to a limited resource. A resource manager will be aware of how much of a specific resource is available, which components are currently using the resource, and how much of the resource the components are using. A resource manager will recommend to policy which components should be pre-empted or resumed based on resource conflicts and availability.~~
- *Policy* is ~~the part of the IL client that sets component group identifiers and priorities based on the current use case responsible for managing component chains or streams. Policy is used to determine if a stream can run based on information including resources, system configuration, and other factors.~~

### **2.17.22.19.2 Example Architecture**

Figure 2-15 shows a high-level architecture diagram of an exemplar OpenMAX IL-based system. In this example, a multimedia framework with a policy manager exists between the applications and the IL layer. This exemplar system also has multiple hardware platforms that are used by different OpenMAX IL components and that are managed by multiple hardware vendor-specific resource managers. But this system would work just as well with a single centralized resource manager.

This example architecture is used as a background for the following discussion on component priorities, behavioral rules and hardware-specific resource managers. It is to be noted, however, that this discussion applies to any OpenMAX IL-based architecture.



**Figure 2-14. Example Architecture**

To ensure consistent component behavior in case of resource conflicts, a common definition of component priority and a set of behavioral rules are needed.

### **2.17.32.19.3 Component Priorities**

Each IL component has a priority value (an OMX\_U32 integer) that the IL client sets.

A descending order of priority is chosen with 0 denoting the highest priority. The following tie-breaking rule also applies: *When comparing components with the same priority, components that have acquired the resource most recently should be deemed to be of higher priority than components that have had the resource longer.*

IL components may also be assigned a group priority by the IL client. Any component sharing the same group ID maintains the same group priority.

### **2.17.42.19.4 Behavioral Rules**

The following behavior is defined on the IL layer:

- The OMX\_ErrorInsufficientResources error is sent by called only on a component that attempts to go to OMX\_StateIdle when there are insufficient resources and sufficient resources cannot be freed by preempting lower priority components.
- A component is not aware that preemption is occurring when it tries to go to OMX\_StateIdle, and the resources it requires need to be freed by preempting lower priority components.
- When a component that has resources which need to be preempted, it will send the OMX\_ErrorResourcesPreempted error to the IL client as it moves from OMX\_StateExecuting or OMX\_StatePause to OMX\_StateIdle. The component will send the OMX\_ErrorResourcesLost error to the IL client as it moves from OMX\_StateIdle to OMX\_StateLoaded once the resources are released.
- In cases where the IL client wants to know when the stream associated with the component can be resumed or started, the IL client shall request to be notified when resources are available. This occurs by putting the component into the OMX\_StateWaitForResources state. When the resources become available, the component automatically goes to OMX\_StateIdle. When the client receives the notification that the component is in OMX\_StateIdle, it can try to move the rest of the components in that chain to OMX\_StateIdle as well. This automatic movement to OMX\_StateIdle ensures that in cases where multiple IL clients are waiting for the same resource, the IL client can resume or start the stream as soon as the resource is available. If the component were to automatically move just to OMX\_StateLoaded, then another IL client could grab that resource first.

These behavioral rules are intended to cover only the interactions between the IL client(s) and the IL components.

#### **2.17.52.19.5      Hardware Vendor-Specific Resource Manager**

To implement the behavioral rules, a hardware vendor-specific resource manager may exist and perform the following functions:

- Implement and manage the wait queue(s).
- Keep track of available resources.
- Keep track of each component that has resources and which resources they are using.
- Notify a component or multiple components that they need to give up their resources when a higher priority component requests the resource.
- Notify the highest priority component waiting for a resource when the resource is available.

The actual interactions between the components and the hardware vendor-specific resource manager(s) are vendor-specific and outside the scope of this document. Section 3 provides more details of the parameter structures and use cases related to priority and resource management.

#### **2.17.62.19.6    Component Suspension**

When a component lacks sufficient resources to process data it may elect to suspend itself as a means to enable more optimal dynamic resource management. Component suspension addresses two use cases:

1. Component has lost an essential resource and the resource loss is potentially temporary in nature.
2. Dynamic allocation of essential resources has failed

In the absence of the ability to suspend, the component's only possible reaction to the preemption and loss of a resource is deinitialization via a transition to OMX\_StateIdle and then OMX\_StateLoaded. Such deinitialization causes the state of the data stream to be lost because the buffers have to be returned to their allocator. Suspension allows a component to retain its state so that it may be resumed at the point of suspension after some delay.

Suspension is a property of a component when it is in OMX\_StateIdle or OMX\_StatePause. Specifically a component is “suspended” when it has lost one or more resources that prevent it from processing data. This means that a component cannot be suspended and be in OMX\_StateExecuting at the same time (since OMX\_StateExecuting implies the component will process or output data whenever that data is available). Therefore, a component may be suspended anytime it is normally holding some resources but not seeking to process data, namely when in OMX\_StateIdle or OMX\_StatePause.

Component suspension requires no new component states but adds one new component-initiated state transition, namely a transition from OMX\_StateExecuting to

`OMX_StatePause` which an executing component performs on itself upon suspension. IL client may perform any of the normal state transitions on a suspended component with the following exception: a client may not transition a suspended component to `OMX_StateExecuting`. Any attempt to do so will fail and return the `OMX_ErrorComponentSuspended` error.

### **2.182.20 Content Pipes**

IL components may leverage content piping to synchronously pull in or push out content (e.g. a filestream) from a source or destination abstracting the platform implementation specifics of the source or destination (e.g. local file, remote file, broadcast, etc). A content pipe is an object that provides content access by implementing the data access abstraction interface defined in the content pipe structure.

The content pipe interface includes functions for conventional content manipulation including:

- opening, closing, and creating content
- seeking to a particular position in the content
- getting the position in the content
- reading data from the current position
- writing data to the current position

~~This content pipe interface also includes functions to accommodate content pipe implementations that may be streaming data asynchronously to or from a remote location. In this case the pipe may not be immediately ready to provide data (in the case of reading) or accept data (in the case of writing). Furthermore such pipes may maintain their own data caches. These functions support:~~

- Checking the pipe for available bytes (either incoming or outgoing) to verify a pipe client may perform a subsequent read or write.
- Reading or writing data via pipe supplied data buffers to avoid unnecessary memory copies between pipe buffers and client buffers.

~~A component that leverages content pipes (e.g. a container demuxer or muxer) acquires the pipe from IL Core via the `OMX_GetContentPipe` function. Alternatively the IL client may provide a custom content pipe (e.g. if the client implements the content pipe itself) via the `OMX_IndexParamCustomContentPipeConfig`. The IL client specifies the target content as a URI via the `OMX_IndexParamContentURI` param.~~

### **2.192.21 File Parsing**

OpenMAX IL defines both standard container format demuxers and the mechanisms to facilitate file parsing functionality in such components. These include means:

- For a component to indicate whether or not it successfully detected and supports the datastream format it was given.

- For a component to inspect and select the streams available on each of the components output ports (when there are multiple alternative streams).
- For the IL client to traverse, extract, and filter the metadata a component captures from a data stream.

## 2.202.22 Video Decoder Error Mapping

A video decoder component has the ability to inform the IL client of any macroblock (MB) errors it encounters while decoding the stream. The client may query the component for a map of the MB errors it has encountered at any time via a dedicated parameter.

One potential use for this functionality is the Video Telephony use case where the video terminal at one end of the connection generates an encoded bitstream for a remote video terminal. The encoded bitstream might get corrupted during transmission resulting in MB errors when the remote terminal receives and decodes it. An application that can communicate with both may extract the MB error map at the decoding terminal and transmit it to the encoding terminal allowing it to refresh the macroblocks in error with intra macroblocks in a subsequent encoded frame.

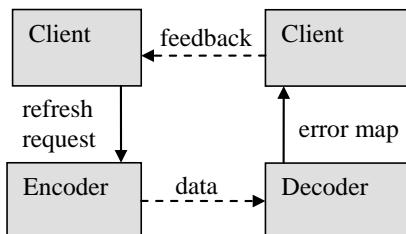


Figure 2-15. Example Use Case for Error Mapping

## 2.212.23 Buffer Payload Additional Information

Depending on buffer payload types and component requirements, a need may arise where additional supporting information will need to be appended to the end of the buffer to further process the buffer payload content within the next component.

For instance, video deblocking algorithms require macroblock level quantization information in order to perform the deblocking process on the video content.

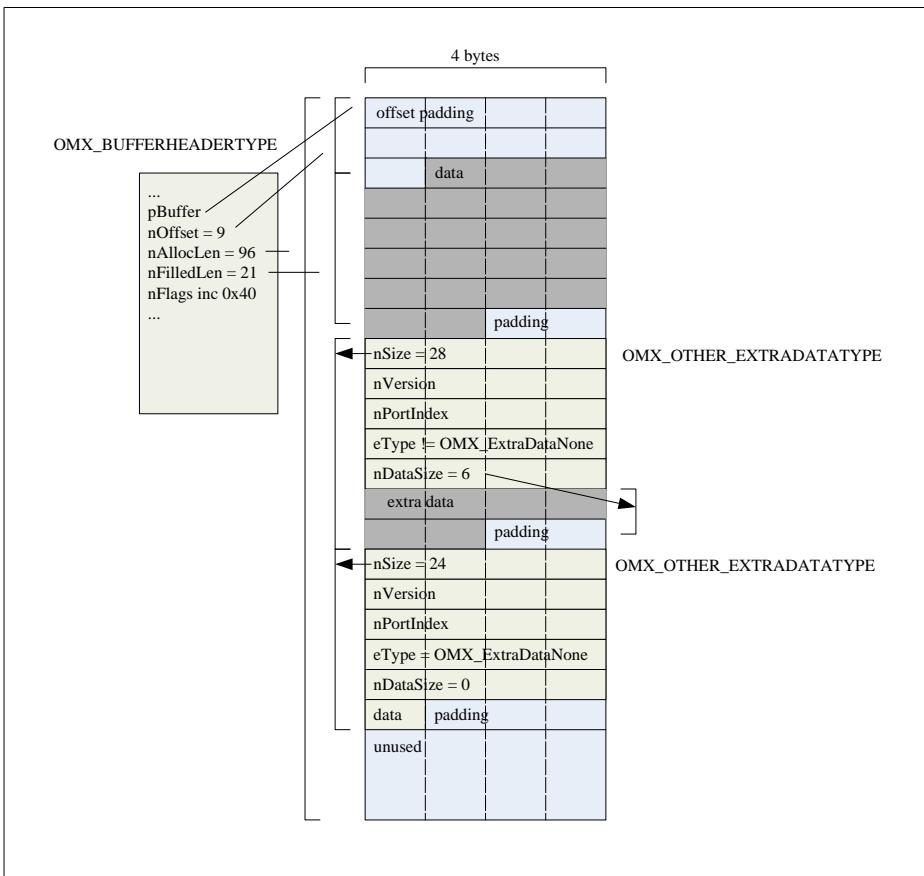
The existence of additional buffer payload information shall be identified via the “extra data” buffer flag within the buffer header structure, which is described in section 3.1.3.7 — `OMX_BUFFERHEADERTYPE`.

This additional buffer payload information applies to the first new logical unit in the buffer. Thus, given the presence of multiple logical units in a buffer, the “extra data” flag applies to the logical unit whose starting boundary occurs first in the buffer. Subsequent

logical units in a buffer don't have explicit "extra data". If explicit "extra data" are required on every logical unit, one or less logical unit should be included in each buffer.

### **2.21.12.23.1 Buffer Data Formatting**

When extra data is present, the data attributes like type and size are identified by a corresponding data ~~structure, immediately~~ structure, immediately following the buffer payload and preceding the actual data. Multiple types of extra data may be appended to the end of the normal payload as series of block pairs (supporting data structure and actual data). To terminate this list of extra data sections, a further data structure should be included in the buffer which indicates that this is the terminating item. For more details see Section 4.2.32.



**Figure 2-16. Formatting of Extra Buffer Data**

## **2.222.24 Endianness**

The endianness used in the implementation of OpenMAX IL API data structures shall obey the endianness of the platform on which the IL client is running. This requirement includes interfaces used by the IL client and interfaces between components (e.g. functions executed exclusively between two tunneling components). The OpenMAX IL implementation is responsible for any endianness conversions inherent in supporting this requirement; any such conversions are transparent to the IL client and to components using the same endianness as the IL client.

### 3 OpenMAX Integration Layer Control API

The OpenMAX Integration Layer API allows integration layer clients to control multimedia components in the audio, video and image domains. An “other” domain is also included to provide for extra functionality, such as audio-video (A/V) synchronization. The user of the OpenMAX Integration Layer API is usually a multimedia framework. In the rest of this document, the user of the OpenMAX Integration Layer API will be referred to as the IL client.

The OpenMAX Integration Layer API is defined in a set of header files, namely:

- `OMX_Types.h`: Data types used in the OpenMAX IL [API](#)
- `OMX_Core.h`: OpenMAX IL core API
- `OMX_Component.h`: OpenMAX IL component API
- `OMX_Audio.h`: OpenMAX IL audio domain data structures
- `OMX_IVCommon.h`: OpenMAX IL structures common to image and video domains
- `OMX_Video.h`: OpenMAX IL video domain data structures
- `OMX_Image.h`: OpenMAX IL image domain data structures
- `OMX_Other.h`: OpenMAX IL other domain data structures (includes A/V synchronization)
- `OMX_Index.h`: Index of all OpenMAX IL-defined data structures
- ~~`OMX_ContentPipe.h`: Content pipe definition~~
- [`OMX\_RoleNames.h`: OpenMAX IL standard role names as defined macros](#)

This section describes how the OpenMAX IL core and OpenMAX IL components are configured for operation.

First, the OpenMAX IL data types are introduced. Next, the methods of the OpenMAX IL core are described. The methods that components implement are discussed in section 3.2.3. Finally, section 3.4 shows calling sequences for a few meaningful operations, including component initialization, normal data flow, data tunnel setup, and data flow in the presence of data tunneling. Such sequence diagrams aim at describing the dynamic interactions between the IL client, the IL core, and the OpenMAX IL components.

When documenting functions, the following convention is used for function parameters:

- `<param_name> [in]` specifies an input parameter, which is set by the function caller and read by the function implementation.
- `<param_name> [out]` specifies an output parameter, which is set by the function implementation and passed back to the caller. When the function returns, the caller can read the new value of the parameter, which is passed as a reference.

- <param\_name> [inout] specifies an input/output parameter, which the function caller can set. The function implementation can modify the parameter before returning it back to the function caller.

This parameter classification can also be found in the OpenMAX IL header files, where the null macros OMX\_IN, OMX\_OUT and OMX\_INOUT are defined. OMX\_IN corresponds to the function parameter <param\_name> [in]. OMX\_OUT corresponds to the function parameter <param\_name> [out], and OMX\_INOUT corresponds to the function parameter <param\_name> [inout].

## 3.1 OpenMAX IL Types

### 3.1.1 *Enumerations*

~~Five 32 bit integer Six~~ enumerations are defined in OMX\_Core.h:

- OMX\_ERRORTYPE is returned by each function defined in the OpenMAX Integration Layer API (see section 3.1.1.3).
- OMX\_COMMANDTYPE includes the possible commands that an IL client can send to an OpenMAX IL component (see section 3.1.1.1).
- OMX\_EVENTTYPE includes events that can be generated inside an OpenMAX IL component and that are passed to the IL client through a callback function (see section 3.1.1.4).
- OMX\_BUFFERSUPPLIERTYPE ~~includes all the possibilities for the buffer supplier in the case of tunneled ports. A description of the use of this enumerative type can be found in~~ section 3.1.1.5.
- OMX\_STATETYPE, which is described in section 3.1.1.2.
- [OMX\\_EXTRADATATYPE](#), which describes the format of extra data carried in data buffers (see section 4.2.32).

Formatted: Font: Times New Roman

#### 3.1.1.1 OMX\_COMMANDTYPE

Table 3-1 represents the possible commands that an IL client can send to an OpenMAX IL component. Since commands are non-blocking, the OpenMAX IL component generates a command completion event via a callback function when the command has completed. Callbacks are defined in a dedicated structure; see section 3.1.3.8.

Table 3-1: OpenMAX IL Commands

Field Name	Description
OMX_CommandStateSet	Change the component state
OMX_CommandFlush	Flush the queue(s) of buffers on a port of a component
OMX_CommandPortDisable	Disable a port on a component
OMX_CommandPortEnable	Enable a port on a component

Field Name	Description
OMX_CommandMarkBuffer	Mark a buffer and specify which other component will raise the event mark received

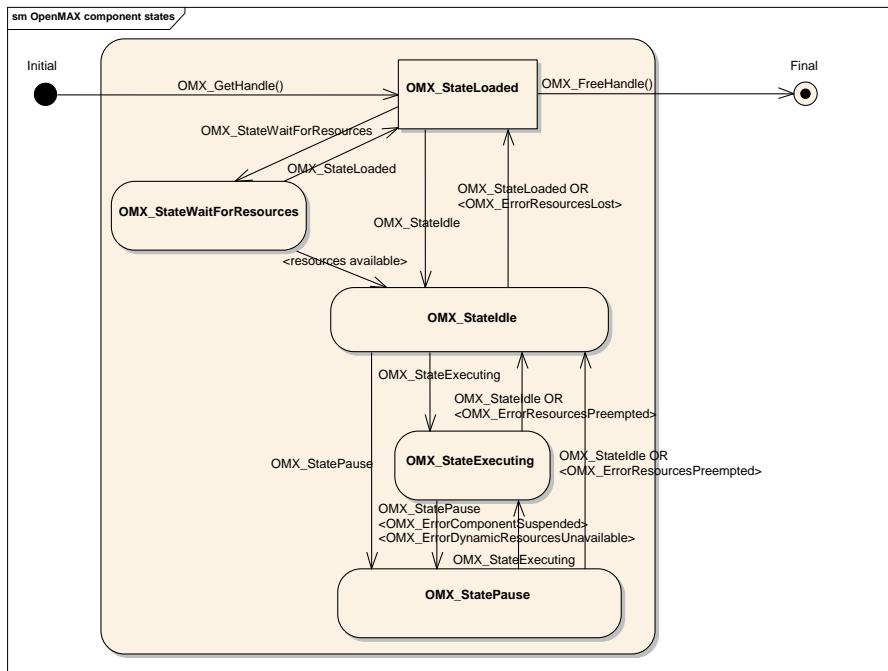
Table 3-2 describes the parameters to be used for each command.

**Table 3-2: Command Syntax**

Command code	nParam	pCmdData
OMX_CommandStateSet	OMX_STATETYPE – state to transition to	NULL
OMX_CommandFlush	OMX_U32 – target port ID	NULL
OMX_CommandPortDisable	OMX_U32 – target port ID	NULL
OMX_CommandPortEnable	OMX_U32 – target port ID	NULL
OMX_CommandMarkBuffer	OMX_U32 – target port ID	OMX_MARKTYPE* - mark data and target component

### 3.1.1.2 OMX\_STATETYPE

Figure 3-1 illustrates the transitions among states. **that** These can occur as a consequence of the IL client calling OMX\_SendCommand(OMX\_CommandStateSet, <state>), where the new state for the component is passed as a parameter. A transition name surrounded by “<” and “>” brackets indicates that the transition is not triggered by a command sent by the IL client but is a consequence of internal component events.



**Figure 3-1. OpenMAX IL Component State Transitions**

This section describes component states. An IL client commands a component to change states via the `OMX_SendCommand` function using the `OMX_CommandStateSet` command.

Table 3-3 represents the states of an OpenMAX IL component.

**Table 3-3: OpenMAX IL Component States**

Field Name	Description	Static Resources Allocated	Location of buffer
<code>OMX_StateInvalid</code>	Component is corrupt or has encountered an error from which it cannot recover.	Unknown	Unknown
<code>OMX_StateLoaded</code>	Component has been loaded but has no resources allocated.	No	Not available
<code>OMX_StateIdle</code>	Component has all <u>static</u> resources but has not transferred any buffers or begun processing data.	Yes	Supplier only
<code>OMX_StateExecuting</code>	Component is transferring buffers and is processing data (if data is available).	Yes	Supplier or non-supplier

<b>Field Name</b>	<b>Description</b>	<b>Static Resources Allocated</b>	<b>Location of buffer</b>
OMX_StatePause	Component data processing has been paused but may be resumed from the point it was paused.	Yes	Supplier or non-supplier
OMX_StateWaitForResources	Component is waiting for a resource to become available.	No	Not available

### 3.1.1.2.1 OMX\_StateLoaded

A component is in the OMX\_StateLoaded state after it has been created via an OMX\_GetHandle call and before allocation of its resources. In this state, the IL client may modify the component's parameters via OMX\_SetParameter, set up data tunnels on the component's ports with OMX\_SetupTunnel, tear down tunnels on the component's port with OMX\_TearDownTunnel, or transition the component to either the OMX\_StateIdle state or the OMX\_StateWaitForResources state.

The IL client may elect to transition a component that is currently in the OMX\_StateLoaded state into the OMX\_StateWaitForResources state if, for example, the component failed to acquire all of its static resources on an attempted transition to the OMX\_StateIdle state.

#### 3.1.1.2.1.1 OMX\_StateLoaded to OMX\_StateIdle

If the IL client requests a state transition from OMX\_StateLoaded to OMX\_StateIdle, the component shall acquire all of its static resources, including buffer headers for all enabled ports, before completing the transition. The component does not acquire buffer headers for any disabled ports. Buffer headers are allocated through calls to OMX\_UseBuffer or OMX\_AllocateBuffer.

The number of buffers and buffer headers used on a port is specified in its port definition (see OMX\_IndexParamPortDefinition), which defaults to the minimum (specified in the same structure) but which may be modified by the supplier before the sequence of OMX\_UseBuffer and OMX\_AllocateBuffer calls via a call to OMX\_SetParameter.

IL allows the actual buffers to be pre-announced during the transition from OMX\_StateLoaded to OMX\_StateIdle. This is achieved by calling OMX\_UseBuffer with a non-NULL pBuffer pointer. This call establishes a one-to-one relationship between the allocated buffer header and the buffer pointer value. No changes to the pointer value or buffer size are allowed until OMX\_FreeBuffer is called on the buffer header. Pre-announcing the buffers may result in components being able to utilize platform specific optimizations for accessing the buffer memory area. Note that pre-announcing buffers matches the behavior mandated in previous versions of the standard (OpenMAX IL 1.1.2 and earlier).

When OMX\_UseBuffer is called with a NULL pBuffer pointer, the actual buffer memory area is available to the non-supplier only during execution, (OMX\_StateExecuting and OMX\_StatePause states) when OMX\_EmptyThisBuffer or OMX\_FillThisBuffer is called. In this case, a buffer allocator port is allowed to change the value of the pBuffer pointer and the nAllocLen field in the buffer header each time the buffer header is passed to the non-supplier. This mode allows flexibility in selecting the buffering scheme.

The caller of the OMX\_UseBuffer macro shall pre-announce either all the buffers or none of the buffers on one port; mixing the two behaviors is prohibited.

For a port connected to the IL client, the IL client may alternatively direct the port to perform the buffer allocations via OMX\_AllocateBuffer calls on the port. For each port, the IL client shall exclusively use OMX\_UseBuffer or OMX\_AllocateBuffer; mixing the two behaviors is prohibited. When OMX\_AllocateBuffer is used, the buffer allocator (i.e., the component) is allowed to change the value of the pBuffer pointer and nAllocLen field in the buffer header each time the buffer header is passed to the IL client.

If the IL client requests a state transition from OMX\_StateLoaded to OMX\_StateIdle, the component shall acquire all of its static resources, including buffers for all enabled ports, before completing the transition. The component does not acquire buffers for any disabled ports. Furthermore, before the transition can complete, the buffer supplier, which is always the IL client when not tunneling, shall ensure that the non-supplier possesses all of its buffers.

For a port connected to the IL client, the IL client may allocate the buffers itself and then pass them to the port via an OMX\_UseBuffer call on the port, or it may direct the port to perform the allocation via an OMX\_AllocateBuffer call on the port. For each port, the IL client shall exclusively use OMX\_UseBuffer or OMX\_AllocateBuffer.

When a port is tunneling, the supplier port either allocates buffers itself or, if the port implements buffer sharing, re-uses buffers from a port on the same component. A tunneling supplier port then passes the buffers to the non-supplier port via an OMX\_UseBuffer call on the non-supplier.

The number of buffers used on a port is specified in its port definition (see OMX\_IndexParamPortDefinition), which defaults to the minimum (specified in the same structure) but which may be modified by the supplier before the sequence of OMX\_UseBuffer and OMX\_AllocateBuffer calls via a call to OMX\_SetParameter.

### **3.1.1.2.2 OMX\_StateIdle**

In the OMX\_StateIdle state, the component is ready to be used, meaning that all necessary static resources have been properly allocated. However, the suppliers retain all their buffers, and no buffer exchange or processing is taking place. Thus, if this state is entered from an OMX\_StateExecuting or OMX\_StatePause state, the component

shall have returned all buffers it was processing to their respective suppliers. The IL client may transition the component to any states other than the ~~OMX\_StateInvalid~~ and OMX\_StateWaitForResources state~~s~~.

### 3.1.1.2.2.1 OMX\_StateIdle to OMX\_StateLoaded

On a transition from OMX\_StateIdle to OMX\_StateLoaded, each buffer supplier shall call OMX\_FreeBuffer on the non-supplier port for each buffer ~~residing at the non-supplier port~~. If the supplier allocated the buffer, it shall free the buffer ~~after~~before calling OMX\_FreeBuffer. If the non-supplier port allocated the buffer, it shall free the buffer upon receipt of an OMX\_FreeBuffer call. Furthermore, a non-supplier port shall always free the buffer header upon receipt of an OMX\_FreeBuffer call. When all of the buffers have been removed from the component, the state transition is complete~~s~~. If the transition was initiated by the component communicates that the initiating OMX\_SendCommand call, the component indicates completion has completed via an OMX\_EventCmdComplete callback event. Alternatively, the component raises an OMX\_ErrorResourcesLost error callback.

### 3.1.1.2.2.2 OMX\_StateIdle to OMX\_StateExecuting

This transition is disallowed when the component is suspended. If the IL client requests a state transition from OMX\_StateIdle to OMX\_StateExecuting and the component is not suspended, the component shall begin transferring and processing data. If the client requests this transition when the component is suspended the component shall ~~fail the call returning raise~~ the OMX\_ErrorComponentSuspended error via the event callback. For ports that communicate with the IL client, the IL client will initiate buffer transfers via OMX\_EmptyThisBuffer and OMX\_FillThisBuffer. Among tunneling ports, any input port that is also a supplier shall transfer its empty buffers to the tunneled output port via OMX\_FillThisBuffer.

### 3.1.1.2.3 OMX\_StateExecuting

In this state, an OpenMAX IL component is transferring and processing data buffers; the component can therefore not be suspended and in this state. The component shall accept calls to OMX\_EmptyThisBuffer on its input ports and OMX\_FillThisBuffer on its output ports. Any port that communicates with the IL client shall call the EmptyBufferDone and FillBufferDone callbacks to return an empty or full buffer, respectively, back to the IL client. Any tunneling port shall call OMX\_FillThisBuffer or OMX\_EmptyThisBuffer on its corresponding tunneled port to return an empty or full buffer, respectively, back to its tunneled port.

In case the buffers were not pre-announced on a port during transition from OMX\_StateLoaded to OMX\_StateIdle or when enabling a port, the buffer pointers and size are allowed to change during execution as described in section 3.1.1.2.1.1. The combined size of buffers placed into a non-supplier port's buffer queue shall never exceed the sum of the nSizeBytes values used on OMX\_AllocateBuffer or

OMX\_UseBuffer calls on the port. This is also the maximum allowable size for an individual buffer.

An IL client may transition a component in the OMX\_StateExecuting state to either the OMX\_StateIdle state or the OMX\_StatePause state.

#### 3.1.1.2.3.1 OMX\_StateExecuting to OMX\_StateIdle

If the IL client requests a state transition from OMX\_StateExecuting to OMX\_StateIdle, the component shall return all buffers to their respective suppliers and receive all buffers belonging to its supplier ports before completing the transition. Any port communicating with the IL client shall return any buffers it is holding via EmptyBufferDone and FillBufferDone callbacks, which are used by input and output ports, respectively. Any non-supplier port shall return all buffers it is holding to the input port or output port it is tunneling with using OMX\_EmptyThisBuffer or OMX\_FillThisBuffer, respectively. Likewise, any supplier tunneling port shall wait for all of its buffers to be returned from its tunneled port.

#### 3.1.1.2.3.2 OMX\_StateExecuting to OMX\_StatePause

A transition from the OMX\_StateExecuting state to the OMX\_StatePause state occurs under ~~in~~ one of ~~three~~ circumstances:

- When the client explicitly requests the transition
- When the component loses a resource required for execution but may be resumed from the point of resource loss if the resource is reacquired later. In this case the component shall execute the transition automatically and issue an error event with the OMX\_ErrorResourcesSuspended OMX\_ErrorComponentSuspended error.
- When the component is unsuccessful in an attempt to allocate dynamic resources. In this case the component shall execute the transition automatically and issue an error event with the OMX\_ErrorDynamicResourcesUnavailable error.
- When a camera component completes a capture and the auto-pause after capture feature has been enabled by the IL client.

#### 3.1.1.2.4 OMX\_StatePause

In this state, an OpenMAX IL component is not transferring or processing data but buffers are not necessarily returned to their suppliers. From the OMX\_StatePause state, execution may be resumed via a transition to OMX\_StateExecuting, preferably without dropping data. However, if the client requests this transition when the component is suspended the component shall fail the call returning the OMX\_ErrorResourcesSuspended error. The component ~~shall~~ ~~may~~ ~~still~~ accept data buffers on all enabled ports at its input, but such buffers will be queued only and not processed ~~further~~. The IL client may transition a component in the OMX\_StatePause state to OMX\_StateIdle or OMX\_StateExecuting. On a transition from

`OMX_StatePause` to `OMX_StateIdle`, the component shall return all buffers to their respective suppliers in a manner identical to the `OMX_StateExecuting`-to-`OMX_StateIdle` transition described in section 3.1.1.2.3.1.

### **3.1.1.2.5 *OMX\_StateWaitForResources***

In this state, the component is waiting for one or more of its required resources to become available. This state is related to resource management. The assumption is that one or more hardware-specific resource managers exist on the platform to handle available resources. The interaction among OpenMAX IL components and resource managers is outside the scope of this specification.

If a component in the `OMX_StateLoaded` state fails to enter the `OMX_StateIdle` state because resources other than buffers are insufficient, the IL client may put the component in the `OMX_StateWaitForResources` state if the IL client wants to be notified when the needed resources become available. The IL client may command the component to discontinue waiting for resources by transitioning it from the `OMX_StateWaitForResources` state to the `OMX_StateLoaded` state. If a component in the `OMX_StateWaitForResources` state acquires all the resources upon which it is waiting, it shall initiate a transition to the `OMX_StateIdle` state.

#### **3.1.1.2.5.1 *OMX\_StateWaitForResources* to *OMX\_StateIdle***

When a component initiates a transition from the `OMX_StateWaitForResources` state to the `OMX_StateIdle` state, it shall communicate the initiation of this transition to the IL client via an `OMX_EventResourcesAcquired` event. When the IL client receives the `OMX_EventResourcesAcquired` event, it shall call `OMX_UseBuffer` and `OMX_AllocateBuffer` in the manner of a transition from `OMX_StateLoaded` to `OMX_StateIdle`. Likewise, the component cannot complete its transition to `OMX_StateIdle` until it acquires all of its static resources, including buffers.

### **3.1.1.2.6 *OMX\_StateInvalid***

*In this state, the component has suffered internal corruption or an error from which it cannot recover. When it detects such a condition, the component transitions itself to `OMX_StateInvalid` and informs the IL client by generating an `OMX_EventError` event with the value `OMX_ErrorInvalidState`. When the IL client receives `OMX_EventError` indicating a transition to `OMX_StateInvalid`, it shall free all resources associated with that component and eventually call `OMX_FreeHandle` to release the handle associated with the component.*

*A component in the `OMX_StateInvalid` state shall fail every call made upon it and return an `OMX_ErrorInvalidState` error message except for `OMX_GetState`, `OMX_FreeBuffer`, or `OMX_ComponentDeinit`. The IL client may also command a transition to the `OMX_StateInvalid` state explicitly via `OMX_SendCommand`. A component may transition between any state and the `OMX_StateInvalid` state.*

### 3.1.1.3 OMX\_ERRORTYPE

The OMX\_ERRORTYPE enumeration shown in Table 3-4 defines the standard OpenMAX IL errors ~~that all functions defined in the OpenMAX IL API return~~. These errors should cover most of the common failure cases. However, vendors are free to add additional error messages of their own as long as they follow these rules:

- Vendor error messages shall be in the range of 0x90000000 to 0x9000FFFF.
- Vendor error messages shall be defined in a header file provided with the component. No error messages are allowed that are not defined.

**Table 3-4: OpenMAX IL Error Codes**

Field Name	Description
OMX_ErrorNone	The function returned successfully.
OMX_ErrorInsufficientResources	There were insufficient resources to perform the requested operation.
OMX_ErrorUndefined	There was an error but the cause of the error could not be determined. <u>When received, the IL client shall treat this error as critical.</u>
OMX_ErrorInvalidComponentName	The component name string was invalid.
OMX_ErrorComponentNotFound	No component with the specified name string was found.
<del>OMX_ErrorInvalidComponent</del>	<del>The component specified did not have a OMX_COMPONENTINITTYPE entry point, or the component did not correctly complete the OMX_COMPONENTINITTYPE call</del>
OMX_ErrorBadParameter	One or more parameters were invalid.
OMX_ErrorNotImplemented	The requested function is not implemented.
OMX_ErrorUnderflow	<del>A buffer with new data was not available when it was needed : e.g. component processing realtime data is not being supplied data in time, the component is underflowing. The buffer was emptied before the next buffer was ready.</del>
OMX_ErrorOverflow	<del>An empty buffer was not available when it was needed, data loss may be occurring : e.g. component processing realtime data did not have an empty buffer to fill with new incoming data, the component is overflowing with data. The buffer was not available when it was needed.</del>
OMX_ErrorHardware	The hardware failed to respond as expected. <u>When received, the IL client shall treat this error as critical.</u>
<del>OMX_ErrorInvalidState</del>	<del>The component is in the OMX_StateInvalid state.</del>

Field Name	Description
<u>OMX_ErrorStreamCorrupt</u>	<p>The stream is found to be corrupt. OMX IL components processing coded data (typically decoders) may have the ability to detect corruption in the data stream. Also they may have the ability to detect missing frames and perform error concealment. Such components should report these errors to the client using this error code on a frame basis. Note that the components <u>shall will in most cases</u> continue normal operation <u>and continue to output data</u>. <u>The port index of the affect port is returned in the nData2 parameter of the EventHandler.</u></p>
<u>OMX_ErrorStreamCorruptStalled</u>	<p><u>The stream is found to be corrupt enough to temporarily stop data output. OMX IL components processing coded data (typically decoders) may have the ability to detect corruption in the data stream. Such components should report these errors to the client using this error code on a frame or buffer basis. The component will stop outputting data but will remain in OMX_StateExecuting and continue to (try to) process the corrupted data. It is entirely possible for the component to resume data output at some point in the future, if the corruption in the stream were to dissipate.</u> This error provides the IL client with the option of either waiting for the stream corruption to dissipate or to take some other action it deems appropriate for that use case. <u>The index of the affected port is returned in the nData2 parameter of the EventHandler.</u></p>
<u>OMX_ErrorStreamCorruptFatal</u>	<p><u>The corruption is such that the component is unable to continue processing the stream even if the corruption were to cease. IL client intervention is required, as the use-case cannot continue. The component shall output no more data on the affected port, but will remain in OMX_StateExecuting. The component shall only report this error once on each port, although the error may be sent again once the port has been disabled or the component transitioned back to loaded state.</u> <u>The index of the affected port is in parameter nData2 of the EventHandler.</u></p>
OMX_ErrorPortsNotCompatible	<p>Ports being set up for tunneled communication are incompatible.</p>

Field Name	Description
OMX_ErrorResourcesLost	Resources allocated to a component in the OMX_StateIdle state have been lost, which has resulted in the component returning to the OMX_StateLoaded state.
OMX_ErrorNoMore	No more indices can be enumerated.
OMX_ErrorVersionMismatch	The component detected a version mismatch.
OMX_ErrorNotReady	The component is not ready to <a href="#">process the call</a> or <a href="#">return data</a> at this time.
OMX_ErrorTimeout	A timeout occurred where the component was unable to process the call in a reasonable amount of time. <a href="#">This could be due to an infinite loop, or busy hardware.</a>
OMX_ErrorSameState	The component tried to transition into the state that it is currently in.
OMX_ErrorResourcesPreempted	Resources allocated to a component in the OMX_StateExecuting or OMX_StatePause states have been preempted, causing the component to return to the OMX_StateIdle state.
<a href="#">OMX_ErrorPortUnresponsive DuringAllocation</a>	<a href="#">The non-supplier port deemed that it had waited an unusually long time for the supplier port to send it an allocated buffer via an OMX_UseBuffer call. A non-supplier port sends this error to the IL client via the EventHandler callback during the allocation of buffers on a transition from OMX_StateLoaded to OMX_StateIdle or on a port enable.</a>
<a href="#">OMX_ErrorPortUnresponsive DuringDeallocation</a>	<a href="#">The non-supplier port deemed that it had waited an unusually long time for the supplier port to request the de-allocation of a buffer header via a OMX_FreeBuffer call. A non-supplier port sends this error to the IL client via the EventHandler callback during the de-allocation of buffers on a transition from OMX_StateIdle to OMX_StateLoaded or on a port disablement.</a>

Field Name	Description
<code>OMX_ErrorPortUnresponsiveDuringStop</code>	The supplier port deemed that it had waited an unusually long time for the non-supplier port to return a buffer via an <code>EmptyThisBuffer</code> or <code>FillThisBuffer</code> call. A supplier port sent this error to the IL client via the <code>EventHandler</code> callback during the disabling of a port, either on a transition from the <code>OMX_StateIdle</code> to <code>OMX_StateLoaded</code> or on a port disablement.
<code>OMX_ErrorIncorrectStateTransition</code>	A state transition was attempted that is not allowed.
<code>OMX_ErrorIncorrectStateOperation</code>	A command or method was attempted that is not allowed during the present state.
<code>OMX_ErrorUnsupportedSetting</code>	One or more values encapsulated in the parameter or configuration structure are unsupported.
<code>OMX_ErrorUnsupportedIndex</code>	The parameter or configuration indicated by the given index is unsupported.
<code>OMX_ErrorBadPortIndex</code>	The port index that was supplied is incorrect.
<code>OMX_ErrorPortUnpopulated</code>	The port has lost one or more of its buffers and is thus unpopulated.
<code>OMX_ErrorComponentSuspended</code>	Component suspended due to temporary loss of resources.
<code>OMX_ErrorDynamicResourcesUnavailable</code>	Component suspended due to inability to acquire dynamic resources.
<code>OMX_ErrorMbErrorsInFrame</code>	Errors detected in frame.
<code>OMX_ErrorFormatNotDetected</code>	OMX IL components performing parsing when reading input buffers or content pipes have the ability to check correct formatting of input data. Such components should report this error to the client (in the form of an <code>OMX_EventError</code> event passed via the <code>EventHandler</code> callback) when it cannot parse or determine the format of the given datastream. This reporting is performed only once in case of file parsing error. In other cases, it is performed on every data unit (e.g. frame) formatting error.
<code>OMX_ErrorContentPipeOpenFailed</code>	Opening the Content Pipe failed
<code>OMX_ErrorContentPipeCreationFailed</code>	Creating the Content Pipe failed

Field Name	Description
OMX_ErrorSeparateTablesUsed	Attempting to query for single Chroma table when separate quantization tables are used for the Chroma (Cb and Cr) coefficients
OMX_ErrorTunnelingUnsupported	Tunneling is not supported by the component
<a href="#">OMX_ErrorInvalidMode</a>	<a href="#">Attempting to apply a setting while in an invalid mode.</a>
<a href="#">OMX_ErrorPortsNotConnected</a>	<a href="#">OMX_TeardownTunnel has been called with two ports that are not currently connected together.</a>
<a href="#">OMX_ErrorContentURINotSpecified</a>	<a href="#">A content URI has not been specified by the IL client. This error is signaled by the component via EventHandler.</a>
<a href="#">OMX_ErrorContentURIError</a>	<a href="#">SetParameter or SetConfig APIs may return this error when the content URI provided is invalid. Alternatively, if an error occurred while accessing the resource identified by a valid content URI, this error shall be signaled to the IL client using the EventHandler callback.</a>
<a href="#">OMX_ErrorCommandCanceled</a>	<a href="#">A command has been canceled.</a>

### 3.1.1.4 OMX\_EVENTTYPE

The OMX\_EVENTTYPE enumeration shown in Table 3-5 includes the event types that an OpenMAX IL component can generate. Section 3.1.3.9 describes events that the OpenMAX IL component generates and passes to the IL client by means of the callback mechanism. Events have associated parameters that are also passed in the callback.

[These are described in detail in Section 3.1.3.9.1.](#)

**Table 3-5: OpenMAX IL Event Types**

Field Name	Description
OMX_EventCmdComplete	Component has completed the execution of a command.
OMX_EventError	Component has detected an error condition.
OMX_EventMark	A buffer mark has reached the target component, and the IL client has received this event with the private data pointer of the mark.
OMX_EventPortSettingsChanged	Component has changed port settings. For example, the component has changed port settings resulting from bit stream parsing.

Field Name	Description
OMX_EventBufferFlag	The event that a component sends when it detects the end of a stream.
OMX_EventResourcesAcquired	The component has been granted resources and is transitioning from OMX_StateWaitForResources to OMX_StateIdle.
OMX_EventComponentResumed	The component has been resumed (i.e. no longer suspended) due to reacquisition of resources.
OMX_EventDynamicResourcesAvailable	The component has acquired previously unavailable dynamic resources.
<u><a href="#">OMX_EventPortFormatDetected</a></u>	<u><a href="#">The component has detected a supported media container format.</a></u>
<u><a href="#">OMX_EventIndexSettingChanged</a></u>	<u><a href="#">The component has reported a settings changed associated with an index on its port.</a></u>
<u><a href="#">OMX_EventPortNeedsDisable</a></u>	<u><a href="#">Component requests the IL client to disable one of its ports.</a></u>
<u><a href="#">OMX_EventPortNeedsFlush</a></u>	<u><a href="#">Component requests the IL client to flush one of its ports.</a></u>

#### 3.1.1.4.1 OMX\_EventCmdComplete

A component generates an OMX\_EventCmdComplete event as soon as a command sent by the IL client has completed its execution, or a component-initiated state transition has occurred. In case of a component state change (whether initiated by the IL client or the component), the new state that the component has entered (or unsuccessfully attempted to enter) is returned as an event parameter. [OMX\\_EventCmdComplete event carries an error code as a parameter to indicate whether the command has completed successfully or not. In case of a successfully completed command, the component returns OMX\\_ErrorNone as a parameter, while in the case of a failure to complete the command, the component returns an appropriate error code value as a parameter. A component that transitions to the OMX\\_StateInvalid state does not generate this event.](#)

#### 3.1.1.4.2 OMX\_EventError

A component generates the OMX\_EventError event when the component detects an error condition; the type of error detected is returned as an event parameter and will use values defined in OMX\_ERRORTYPE. A component shall send the following errors via OMX\_EventError:

- A component sends the OMX\_ErrorInvalidState error if the component transitions to the OMX\_StateInvalid state.
- A component sends the OMX\_ErrorResourcesPreempted error if the component transitions from OMX\_StateExecuting or OMX\_StatePause to OMX\_StateIdle due to the loss of a resource.
- A component sends the OMX\_ErrorResourcesLost error if the component transitions from OMX\_StateIdle to OMX\_StateLoaded due to the loss of a resource.

#### **3.1.1.4.3 OMX\_EventMark**

A component may generates the OMX\_EventMark event when it receives a marked buffer. When a component receives a buffer, it shall compare its own pointer to the pMarkTargetComponent field contained in the buffer. If the pointers match, then the component shall send a mark event including pMarkData as a parameter, immediately after the component has finished processing the buffer. The IL client can use the mark event to measure the propagation delay of a data buffer through a chain of components, or to notify a component that a particular buffer has reached the given destination.

#### **3.1.1.4.4 OMX\_EventPortSettingsChanged**

A component generates the OMX\_EventPortSettingsChanged event as soon as component port settings change. For example, a video decoder may not know *a priori* the output frame size and frame rate, as these parameters are coded in the input bit stream. As soon as such parameters are parsed, the component changes the values of the configuration structures of its output port and sends the OMX\_EventPortSettingsChanged event to the IL client. If a settings change requires the IL client to either reallocate buffers or recycle the tunnel on the port that generated the OMX\_EventPortSettingsChanged then that port shall cease transferring data until the IL client takes such action. A component generates the OMX\_EventPortSettingsChanged event after it changes the values of the configuration structures of its port, the event is sent to the IL client. Upon receiving the event notification, the IL client may use OMX\_GetParameter() or OMX\_GetConfig() – whichever is appropriate – to retrieve the updated port settings.

This event may be issued within any component state and from ports that are disabled.

A component may decide to change its configuration structures associated with its port for a variety of reasons:

- During the course of processing a stream, it may discover new embedded stream properties within in.

For example, a video decoder may discover new frame size and frame rate embedded within the bit stream it is processing.

- Some components require port settings that are common between their input and output ports. These components are not able to perform any conversions between these common settings, an attempt by an IL client to apply a configuration setting on the input port that differs from the current setting on the output port may trigger an immediate update of the output port settings to ensure the common settings are always maintained between the ports.

For example, attempting to apply a new sample rate setting on an audio decoder input port that is not capable of performing sample rate conversion between its output port shall trigger an output port settings change.

When an OMX\_EventPortSettingsChanged event is emitted from a component's output port, the component shall cease transferring data on that port until the IL client takes an action to commence it again. Ceasing the transfer of buffers is required in order to coordinate the updated port settings with any downstream component and possibly facilitate the re-allocation of new port resources (e.g. port buffers). In order to commence the emission of data again on the output port, the IL client shall disable and re-enable the port – this action will also give the component port the opportunity to reallocate any new buffer requirements associated with the port settings change between the IL client or its tunneled port.

In cases when an input port emits the event and the port settings changed is associated with OMX\_IndexParamPortDefinition, the component may need to have the port disabled and re-enabled in order to reallocate any new buffer requirements associated with the settings change. To prevent the loss of any input data, the component issuing the OMX\_EventPortSettingsChanged event on its input port should buffer all input port data that arrives between the emission of the OMX\_EventPortSettingsChanged event and the arrival of the command to disable the input port. For all other parameter indexes reported via the OMX\_EventPortSettingsChanged event for an input port, the IL client is not required to disable and re-enable the port(s).

### **3.1.1.4.5 OMX\_EventBufferFlag**

A component generates the OMX\_EventBufferFlag event when an output port emits a buffer with the OMX\_BUFFERFLAG\_EOS flag set in the nFlags field. The nData1 field of EventHandler specifies the value of the output port's portindex field. The nData2 field of EventHandler specifies the unaltered nFlags field containing the end-of-stream (EOS) flag.

If a component does not propagate a stream further (e.g., the component is an audio or video sink), then the component shall send an OMX\_EventBufferFlag event for that stream when it has finished processing a buffer with OMX\_BUFFERFLAG\_EOS set. The nData1 field of EventHandler specifies the input port that received the buffer. The nData2 field of EventHandler specifies the unaltered nFlags field containing the EOS flag.

#### ***3.1.1.4.6 OMX\_EventResourcesAcquired***

A component generates the OMX\_EventResourcesAcquired event when it is in the OMX\_StateWaitForResources state, and the resource manager detects that the needed resources are available. When the component generates this event, it is ready to change state into the OMX\_StateIdle, *and it waits for all the one* buffers *to be are* allocated and assigned to its ports.

#### ***3.1.1.4.7 OMX\_EventComponentResumed***

A suspended component generates the OMX\_EventComponentResumed event when the resources it had lost have been reacquired. Upon receipt of this event the component is no longer suspended *and the* client may attempt to transition a suspended component into OMX\_StateExecuting.

#### ***3.1.1.4.8 OMX\_EventDynamicResourcesAvailable***

A suspended component generates the OMX\_EventDynamicResourcesAvailable event when some dynamic resource it was formerly unable to allocate has become available. Upon receipt of this event the component is no longer suspended and the client may attempt to transition it into OMX\_StateExecuting.

#### ***3.1.1.4.9 OMX\_EventPortFormatDetected***

*A component, such as a media container demuxer, generates the OMX\_EventPortFormatDetected event when it detects the content format. By issuing this event the component informs that IL client that it is able to process the content. An example of this event usage is available in section 3.4.6.*

#### ***3.1.1.4.10 OMX\_EventIndexSettingChanged***

*A component generates the OMX\_EventIndexSettingChanged event when it detects parameter or config setting change, if the client has previously requested notifications for this index using the controls defined in Section 4.6.4. Upon receipt of this event the IL client may use OMX\_GetParameter() or OMX\_GetConfig() as appropriate to retrieve the new setting value.*

### **3.1.1.4.11 OMX\_EventPortNeedsDisable**

A component generates the `OMX_EventPortNeedsDisable` event when it is processing a disable port command on a port doing buffer sharing and it needs the IL client to disable the sharing port in order to be able to complete the current disable port command.

### **3.1.1.4.12 OMX\_EventPortNeedsFlush**

A component generates the `OMX_EventPortNeedsFlush` event when it is processing a flush command on a port doing buffer sharing and it needs the IL client to flush the sharing port in order to be able to complete the current flush command.

## **3.1.1.5 OMX\_BUFFERSUPPLIERTYPE**

The `OMX_BUFFERSUPPLIERTYPE` [enumerative type](#) shown in Table 3-6 specifies the port in the tunnel that is the supplier port. A buffer supplier port either may allocate its buffers or reuse buffers provided by another port within the same component.

**Table 3-6: OpenMAX IL Buffer Supplier Type For Tunnel Setup**

Field Name	Description
<code>OMX_BufferSupplyUnspecified</code>	The port supplying the buffers is unspecified, or no supplier is preferred.
<code>OMX_BufferSupplyInput</code>	The input port supplies the buffers.
<code>OMX_BufferSupplyOutput</code>	The output port supplies the buffer.

## **3.1.2 Data Types**

Table 3-7 identifies the data types available in the specification.

**Table 3-7: OpenMAX IL Data Types**

Data Type	Description
<code>OMX_U8</code>	An 8 bit unsigned quantity that is byte aligned.
<code>OMX_S8</code>	An 8 bit signed quantity that is byte aligned.
<code>OMX_U16</code>	A 16 bit unsigned quantity that is 16 bit word aligned.
<code>OMX_S16</code>	A 16 bit signed quantity that is 16 bit word aligned.
<code>OMX_U32</code>	A 32 bit unsigned quantity that is 32 bit word aligned.
<code>OMX_S32</code>	A 32 bit signed quantity that is 32 bit word aligned.
<code>OMX_U64</code>	A 64 bit unsigned quantity that is 64 bit word aligned.
<code>OMX_S64</code>	A 64 bit signed quantity that is 64 bit word aligned.

<u>Data Type</u>	<u>Description</u>
<u>OMX_BU32</u>	<p>A bounded 32 bit unsigned quantity.</p> <pre>typedef struct OMX_BU32 {     OMX_U32 nValue;     OMX_U32 nMin;     OMX_U32 nMax; } OMX_BU32;</pre> <p><u>nValue</u> represents the actual value;  <u>nMin</u> represents the minimum for the value (i.e. <u>nValue &gt;= nMin</u>)  <u>nMax</u> represents the maximum for the value (i.e. <u>nValue &lt;= nMax</u>)</p>
<u>OMX_BS32</u>	<p>A bounded 32 bit signed quantity.</p> <pre>typedef struct OMX_BS32 {     OMX_S32 nValue;     OMX_S32 nMin;     OMX_S32 nMax; } OMX_BS32;</pre> <p><u>nValue</u> represents the actual value;  <u>nMin</u> represents the minimum for the value (i.e. <u>nValue &gt;= nMin</u>)  <u>nMax</u> represents the maximum for the value (i.e. <u>nValue &lt;= nMax</u>)</p>
<u>OMX_BOOL</u>	Represents a true (OMX_TRUE) or false (OMX_FALSE) value.
<u>OMX_PTR</u>	Used to pass pointers between the OMX applications and the OMX Core and components.
<u>OMX_STRING</u>	Used to pass "C" type strings between the application and the core and component. This is a pointer to a zero terminated string.
<u>OMX_UUIDTYPE</u>	A very long unique identifier to uniquely identify at runtime. This identifier should be generated by a component in a way that guarantees that every instance of the identifier running on the system is unique.
<u>OMX_ALL</u>	Used to as a wildcard to select all entities of the same type when specifying the index, or referring to an object by an index. (i.e. use OMX_ALL to indicate all N channels). When used as a port index for a config, or parameter or command this OMX_ALL denotes that the config, or parameter or command applies to the entire component not just one port/all ports on the component.
<u>OMX_TICKS</u>	Used to represent time or duration in microseconds. Refer to 6.2.1 for more information.

<u>Data Type</u>	<u>Description</u>
<u>OMX_HANDLETYPE</u>	Defines the component handle as seen by the IL client. The component handle is used by the IL core to access all of the public methods of the component. The component handle also contains pointers to the private data area of the component. The OpenMAX IL core allocates and initializes the component handle with help from the component during the process of loading the component. After the component is successfully loaded, the IL client can safely access any of the public functions of the component via the IL core macros, although some may return an error because the state of the component is inappropriate for the access. The component handle shall not be used directly by the IL client or other IL components to access the public methods of an implementation. The IL core macros shall be used instead. Represents a component handle. Refer to 3.1.5 for more information.

### 3.1.23.1.3 Structures

This section discusses the data structures defined in the OpenMAX IL core. Data-specific structures are discussed in Section 4. The first two fields of each OpenMAX IL data structure denote the size, nSize, of the structure and the version of type OMX\_VERSIONTYPE, nVersion, which is defined in section 3.1.3.3. The entity that allocates an OpenMAX IL structure is responsible for filling in these two values. Hereinafter, definitions for these two common fields are omitted in individual structure definitions.

Table 3-8 lists structures that are associated with an index.

Table 3-8: Indices with their associated structures

<u>OpenMAX IL Indices ( OMX_Index.h )</u>	<u>Corresponding OpenMAX IL Structures</u>
<u>OMX_IndexParamPriorityMgmt</u>	<u>OMX_PRIORITYMGMTTYPE</u>
<u>OMX_IndexParamDisableResourceConcealment</u>	<u>OMX_RESOURCECONCEALMENTTYPE</u>
<u>OMX_IndexParamSuspensionPolicy</u>	<u>OMX_PARAM_SUSPENSIONPOLICYTYPE</u>
<u>OMX_IndexParamComponentSuspended</u>	<u>OMX_PARAM_SUSPENSIONTYPE</u>
<u>OMX_IndexParamAudioInit</u>	<u>OMX_PORT_PARAM_TYPE</u>
<u>OMX_IndexParamImageInit</u>	<u>OMX_PORT_PARAM_TYPE</u>
<u>OMX_IndexParamVideoInit</u>	<u>OMX_PORT_PARAM_TYPE</u>
<u>OMX_IndexParamOtherInit</u>	<u>OMX_PORT_PARAM_TYPE</u>
<u>OMX_IndexParamCompBufferSupplier</u>	<u>OMX_PARAM_BUFFERSUPPLIERTYPE</u>
<u>OMX_IndexParamPortDefinition</u>	<u>OMX_PARAM_PORTDEFINITIONTYPE</u>

<u>OpenMAX IL Indices ( OMX_Index.h )</u>	<u>Corresponding OpenMAX IL Structures</u>
<u>OMX_IndexConfigTunneledPortStatus</u>	<u>OMX_CONFIG_TUNNELEDPORTSTATUSTYPE</u>

### **3.1.2.13.1.3.1 OMX\_COMPONENTREGISTERTYPE**

The OMX\_COMPONENTREGISTERTYPE structure is used in the case of static linking of components to the core. The core optionally uses it to load the component and run the specific component initialization functions.

OMX\_COMPONENTREGISTERTYPE is defined as follows.

```
typedef struct OMX_COMPONENTREGISTERTYPE
{
    const char          * pName;
    OMX_COMPONENTINITTYPE pInitialize;
} OMX_COMPONENTREGISTERTYPE;
```

#### **3.1.3.1.1 Parameter Definitions**

- pName contains the string name of the component and has limit of 128 bytes (including '\0').
- pInitialize contains the pointer to the initialization function of the component. The OMX\_COMPONENTINITTYPE type definition is the type of function pointer for the component initialization entry point. The definition is as follows:

### **3.1.2.2 OMX\_COMPONENTINITTYPE Type Definition**

The OMX\_COMPONENTINITTYPE type definition is the type of function pointer for the component initialization entry point. The definition is as follows:

```
typedef OMX_ERRORTYPE (* OMX_COMPONENTINITTYPE)(OMX_IN OMX_HANDLETYPE hComponent);
```

#### **3.1.2.2.1 Parameter Definitions**

- pName contains the string name of the component and has limit of 128 bytes (including '\0').
- pInitialize contains the pointer to the initialization function of the component.

### **3.1.2.33.1.3.2 OMX\_ComponentRegistered[ ]**

Any core that statically links its components shall define this global array containing the list of all registered components in the form of OMX\_COMPONENTREGISTERTYPE fields.

### **3.1.2.43.1.3.3 OMX\_VERSIONTYPE**

The OMX\_VERSIONTYPE type indicates the version of a component or structure. Each structure uses an OMX\_VERSIONTYPE field to indicate the OpenMAX IL specification version under which the structure is defined. For OpenMAX IL version 1.0, the specification version is 1.0.R.S with any Revision R and Step S values. For OpenMAX IL version 1.24, the specification version is 1.24.R.S with any Revision R and Step S values. The component structure also includes an OMX\_VERSIONTYPE field to indicate a vendor-specific component version.

OMX\_VERSIONTYPE is defined as follows.

```
typedef union OMX_VERSIONTYPE
{
    struct
    {
        OMX_U8 nVersionMajor;
        OMX_U8 nVersionMinor;
        OMX_U8 nRevision;
        OMX_U8 nStep;
    } s;
    OMX_U32 nVersion;

} OMX_VERSIONTYPE;
```

### **3.1.2.4.13.1.3.3.1 Parameter Definitions**

- nVersionMajor identifies the major version number. This byte of the version occurs first.
- nVersionMinor identifies the minor version number.
- nRevision identifies the revision number.
- nStep identifies the step number. This byte of the version occurs last.
- nVersion is an alternative way of accessing the version information.

OMX\_VERSION\_MAJOR, OMX\_VERSION\_MINOR, OMX\_VERSION\_REVISION,  
and OMX\_VERSION\_STEP and OMX\_VERSION defines are available to identify the  
specification version information.

### **3.1.2.53.1.3.4 OMX\_PRIORITYMGMTTYPE**

The IL client may use the OMX\_IndexConfigPriorityMgmt and OMX\_IndexParamPriorityMgmt parameters with the OMX\_PRIORITYMGMTTYPE structure. This structure describes the priority assigned to a set of components. A component group identifies a set of co-dependent components associated with the same feature. All components in the same group share the same group ID and priority. If one component in a group loses resources and stops running, the entire feature they collectively contribute to is lost. In this case, the [IL client](#) should transition all of the other components in the same group to OMX\_StateLoaded. A component that is the only one with a certain nGroupID acts atomically.

OMX\_PRIORITYMGMTTYPE is defined as follows.

```
typedef struct OMX_PRIORITYMGMTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nGroupPriority;
    OMX_U32 nGroupID;
} OMX_PRIORITYMGMTTYPE;
```

#### **3.1.2.5.13.1.3.4.1 Parameter Definitions**

- nGroupPriority is the priority value associated with a group of components. If a parameter of this type is assigned to a component, that component belongs to the group identified with nGroupID and has a priority equal to nGroupPriority. By definition, the value 0 represents the highest priority for a group of components.  
The exact mechanism to assign priorities to groups of components is outside the scope of this document.  
The group is treated as having the same priority. When the priority of one component in the group is changed, that change effectively applies to all components in the group. The [IL client](#) shall update each component's priority within the group with the same priority. The suspension of one component in a group does not imply the suspension of all components in that group.
- nGroupID is a unique ID for all components in the same component group.

### **3.1.2.63.1.3.5 OMX\_RESOURCECONCEALMENTTYPE**

The IL client may use the OMX\_IndexParamDisableResourceConcealment parameter with the OMX\_RESOURCECONCEALMENTTYPE structure to enable or disable resource concealment in a component.

The definition of OMX\_RESOURCECONCEALMENTTYPE is shown as follows:

```
typedef struct OMX_RESOURCECONCEALMENTTYPE
{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
```

```

    OMX_BOOL bResourceConcealmentForbidden;
} OMX_RESOURCECONCEALMENTTYPE;

```

### 3.1.2.6.13.1.3.5.1 Parameter Definitions

- bResourceConcealmentForbidden is a Boolean that shall disallow the use of resource concealment methods by a component to resolve resource conflicts.

### 3.1.2.73.1.3.6 Component Suspension Policy

A component lacking sufficient resources to process data may elect to suspend itself to resolve a temporary resource conflict. Component suspension is ideal when the resource loss is temporary in nature or driven by a requirement for additional runtime dynamic resources.

The IL client specifies the suspension policy of a component via a parameter, OMX\_IndexParamSuspensionPolicy, where possible suspension policies include:

- Suspension Disabled: The component shall not suspend itself. If a component in OMX\_StateExecuting loses resource it shall transition through OMX\_StateIdle, into OMX\_StateLoaded as part of its resource loss. This shall be the **default** component behavior as defined in v1.0.
- Suspension Enabled: Upon detection of a temporary loss of resources a component may suspend processing. No state transitions are triggered if suspension occurs in OMX\_StatePause or OMX\_StateIdle. If the component is in OMX\_StateExecuting when it suspends, it shall transition to OMX\_StatePause.

The OMX\_PARAM\_SUSPENSIONPOLICYTYPE is defined as follows:

```

typedef struct OMX_PARAM_SUSPENSIONPOLICYTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_SUSPENSIONPOLICYTYPE ePolicy;
} OMX_PARAM_SUSPENSIONPOLICYTYPE;

```

The parameters for OMX\_PARAM\_SUSPENSIONPOLICYTYPE are defined as follows.

- ePolicy specifies to the component whether suspension is enabled or disabled, the default value to support suspension, OMX\_SuspensionEnabled, or to disable support for suspension, OMX\_SuspensionDisabled. The component **default** shall be OMX\_SuspensionDisabled.

Table 3-9: OMX\_SUSPENSIONPOLICYTYPE enumeration

Field Name	Description
<u>OMX_SuspensionDisabled</u>	<u>Suspension disabled</u>
<u>OMX_SuspensionEnabled</u>	<u>Suspension enabled</u>

An IL client may query if the component is suspended using the OMX\_IndexParamComponentSuspended parameter. The client can use this suspension status of the component to make decisions on how to proceed when a component is suspended. The IL client may opt to leave the component as-is expecting the suspension to be temporary. The IL client may opt to transition the component to the loaded state, or perform some alternative processing.

The OMX\_PARAM\_SUSPENSIONTYPE is defined as follows:

```
typedef struct OMX_PARAM_SUSPENSIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_SUSPENSIONTYPE eType;
} OMX_PARAM_SUSPENSIONTYPE;
```

The parameters for OMX\_PARAM\_SUSPENSIONTYPE are defined as follows.

- eType specifies the suspension state of the component where OMX\_Suspended indicates suspension and OMX\_NotSuspended is the converse.

**Table 3-10: OMX\_SUSPENSIONTYPE enumeration**

<u>Field Name</u>	<u>Description</u>
<u>OMX_NotSuspended</u>	<u>Component is not suspended</u>
<u>OMX_Suspended</u>	<u>Component is suspended</u>

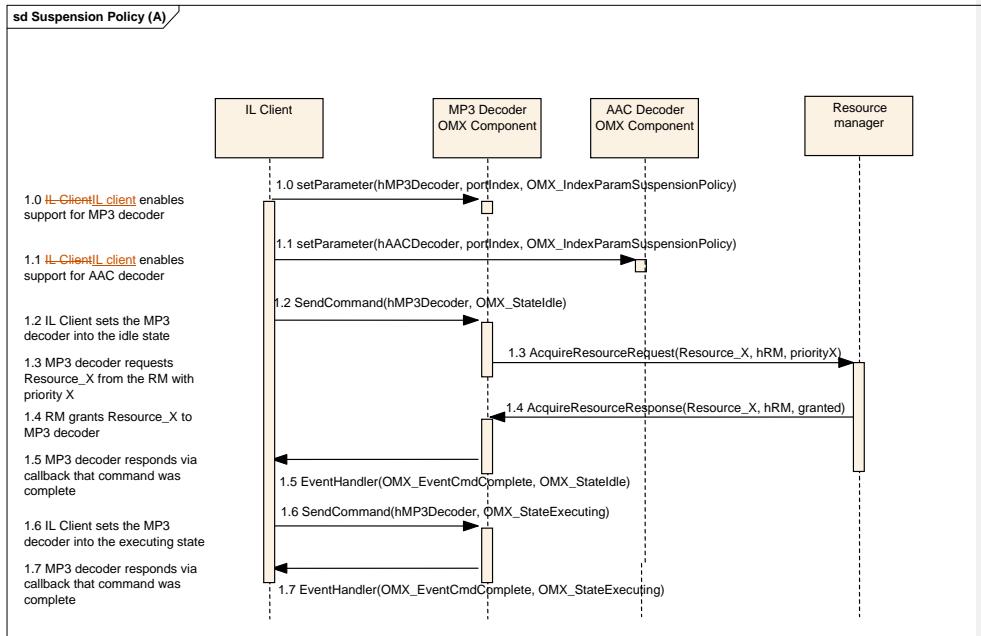
### **3.1.2.7.13.1.3.6.1 Suspension Due to Pre-emption of Resources**

The effect of “suspension” on component implementations is minimal, specifically:

- Upon the loss of one or more resources, a component shall decide between either suspending itself (if it is capable of resumption later and its suspension policy allows it) or de-initializing itself via OMX\_ErrorResourcesPreempted/Lost (if it is incapable of resumption later or if its suspension policy disallows suspension).
- In the case of suspension the component shall send the OMX\_ErrorComponentSuspended error to the IL client. If the component is in OMX\_StateExecuting the component shall transition itself to OMX\_StatePause and send the OMX\_EventCmdComplete event to the IL client.
- If the component supports suspension, the component shall support the OMX\_IndexParamComponentSuspended parameter.
- Upon a request to transition to Executing the component shall validate that it is not suspended. If it is suspended, the component shall fail the transition with an OMX\_ErrorComponentSuspended error.
- Upon reacquisition of resources the component signals the IL client via the OMX\_EventComponentResumed event. The component remains in

OMX\_StatePause until the IL client resumes the component by transitioning it back to OMX\_StateExecuting.

Upon the de-allocation of resources, the component shall be aware of which resources have already been de-allocated from a suspension.



**Figure 3-2: Suspension Policy (A)**

**Figure 3-3: Suspension Policy (B)**

#### **Figure 3-4: Component Suspension Due to Pre-emption of Resources**

Figure 3-2, Figure 3-3, and Figure 3-4 comprise an example of two components, MP3 decoder and AAC decoder, requiring access to a common resource. Assume that each component needs to process a set of compressed buffers to be decoded. The IL client sets the components to support the suspension mechanism (1.0 A and 1.1 A) so that any loss of resources while processing the streams can be resumed.

The IL client transitions the MP3 decoder into OMX\_StateIdle (1.2 A). At this time the MP3 decoder issues a request to the resource manager (RM) for Resource\_X (1.3 A). The RM responds to the request by granting Resource\_X to the MP3 decoder (1.4 A). The MP3 decoder is then transitioned to start processing of stream buffers. (Note the buffer transfers are not shown in the diagram for simplicity).

Next the IL client transitions the AAC decoder into OMX\_StateIdle (1.0 B). The AAC decoder issues a request for Resource\_X with a higher priority client to the RM (1.1 B). The RM in turn issues a request to the MP3 decoder to release Resource\_X (1.2 B). The MP3 decoder complies and releases Resource\_X to the RM (1.3 B).

The MP3 decoder at this point sends an error to the IL client to indicate that the component is suspended (1.4 B). The MP3 decoder issues an acquire resource request for Resource\_x (1.5 B) which of course the RM cannot fulfill since it is a lower priority request but the RM will track this resource request for the MP3 decoder.

~~The next step for the MP3 decoder is to transition to OMX\_StatePause (1.6 B) and then emit a command complete paused event to the IL client (1.7 B). At this point the MP3 decoder is in a paused suspension state.~~

~~Concurrently, the RM may also grant Resource\_X to the AAC decoder after being released by the MP3 decoder (1.7 B). The AAC decoder completes the state change to OMX\_StateIdle by issuing a command complete to the IL client. Assuming the IL client transitions the AAC decoder to executing and after processing a number of buffers (1.0 C) the AAC decoder releases Resource\_X (1.1 C).~~

~~The RM then grants Resource\_X to the MP3 component (1.2 C) base on its earlier request (1.5 B). The MP3 decoder then sets its suspension type to resume (1.3 C) and then issues an OMX\_EventComponentResumed message to the IL client (1.4 C). The IL client transitions the MP3 component out of OMX\_StatePause to OMX\_StateExecuting to resume the stream processing (1.5 C 1.6 C).~~

### **3.1.2.7.23.1.3.6.2 Suspension Due to Unavailable Dynamic Resources**

Under certain conditions the size and type of component resources vary within the lifetime of the component. As an example, resource requirements are dependent upon properties of the data stream itself, which are known only after inspection of the stream. This implies a component is in the executing state by which point all static resources shall be allocated.

A component in the executing state may attempt to allocate additional dynamic resources as a result of increased requirements during processing. This dynamic resource allocation is completely transparent to the client except in the case where the component fails to allocate resources while in OMX\_StateExecuting. Upon failure to allocate resources the component issues an error, OMX\_ErrorDynamicResourcesUnavailable, and transitions to OMX\_StatePause if the component suspension policy has been previously enabled by the IL client.

The component upon receiving the dynamic resources issues the event OMX\_EventDynamicResourcesAvailable to the IL client and remains in OMX\_StatePause. The component remains in OMX\_StatePause until the IL client resumes the component by transitioning it back to OMX\_StateExecuting.

The suspension mechanism follows the case where suspension occurs as a result of preemption with the exception of the errors and events presented to the IL client.

### **3.1.2.83.1.3.7 OMX\_BUFFERHEADERTYPE**

~~In the context of a single port, each data buffer has a header associated with it that contains meta-information about the buffer. The IL client shares buffer headers with each port with which it is communicating. Likewise, each-a connected pair of tunneled ports share buffer headers. If a data buffer is shared across multiple tunnels, each tunnel uses a distinct set of buffer headers.; otherwise, the same buffer transferred over multiple ports will have distinct buffer headers associated with it for each port.~~

The definition of the buffer header is shown as follows.

```
typedef struct OMX_BUFFERHEADERTYPE
{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U8* pBuffer;
    OMX_U32 nAllocLen;
    OMX_U32 nFilledLen;
    OMX_U32 nOffset;
    OMX_PTR pAppPrivate;
    OMX_PTR pPlatformPrivate;
    OMX_PTR pInputPortPrivate;
    OMX_PTR pOutputPortPrivate;
    OMX_HANDLETYPE hMarkTargetComponent;
    OMX_PTR pMarkData;
    OMX_U32 nTickCount;
    OMX_TICKS nTimeStamp;
    OMX_U32 nFlags;
    OMX_U32 nOutputPortIndex;
    OMX_U32 nInputPortIndex;
} OMX_BUFFERHEADERTYPE;
```

### **3.1.2.8.13.1.3.7.1 Parameter Definitions**

- `pBuffer` is a pointer to the actual buffer where data is stored but not necessarily the start of valid data; for more information, see the description of `nOffset` below.
- `nAllocLen` is the total size of the allocated buffer in bytes, including valid and unused bytes. If `pBuffer` is updated, this field shall be updated accordingly.
- `nFilledLen` is the total size of valid bytes currently in the buffer starting from the location specified by `pBuffer` and `nOffset`. This includes any padding, e.g. the unused bytes at the end of a line of video when stride in bytes is larger than width in bytes.
- `nOffset` is the start offset of valid data in bytes from the start of the buffer. The value of `nOffset` is set by the entity that places data into the buffer (or forwards data in a shared buffer). A pointer to the valid data may be obtained by adding `nOffset` to `pBuffer`.
- `pAppPrivate` is a pointer to an IL client private structure.
- `pPlatformPrivate` is a pointer to a private platform-specific structure. For instance, in the case where the IL client allocates the buffer through the platform's memory manager, this structure may contain information the platform's memory manager associates with the buffer.

- `pOutputPortPrivate` is a private pointer of the output port that uses the buffer. If a buffer header is used on an input port communicating with the IL client, the value of the buffer's `pOutputPortPrivate` is undefined.
- `pInputPortPrivate` is a private pointer of the input port that uses the buffer. If a buffer header is used on an output port communicating with the IL client, the value of the buffer's `pInputPortPrivate` is undefined.
- `hMarkTargetComponent` is the handle of the component that should emit an `OMX_EventMark` event upon processing this buffer. A NULL handle indicates that the buffer carries no mark. The `OMX_CommandMarkBuffer` command provides this handle to the marking component. The marking component, in turn, copies this handle to the marked buffer. Each component that is processing a buffer ~~shall~~ compare its own handle to this handle and if they match it shall emit the mark using the `OMX_EventMark` event, and remove the mark from the buffer. emit the mark if the handles match. If the handle of the processing component does not match this field theA component shall propagate~~old hMarkTargetComponent~~ propagate this field from an input buffer to its associated output buffer.
- The `pMarkData` pointer refers to IL client-specific data associated with the mark that is sent on `OMX_EventMark` when emitted. Upon receipt of a mark, the IL client may use this data to disambiguate this mark from others. The `OMX_CommandMarkBuffer` command provides this pointer to the marking component. The marking component, in turn, copies this pointer to the marked buffer. A component ~~shall~~ propagate this field from an input buffer to its associated output buffer, unless the mark has matched the component handle and the `pMarkData` has been returned in the `OMX_EventMark` event.
- `nTickCount` is an optional entry that the component and IL client can update with a tick count when they access the component; not all components will update it. The value of `nTickCount` is in microseconds. Since this is a value relative to an arbitrary starting point, `nTickCount` cannot be used to determine absolute time.
- `nTimeStamp` is a-the presentation timestamp corresponding to the sample starting at the first logical sample boundary in the buffer. Timestamps of successive samples within the buffer may be inferred by adding the duration of the preceding buffer to the timestamp of the preceding buffer. A component should propagate this field from an input buffer to its associated output buffer.
- `nFlags` field contains buffer specific flags, such as the EOS flag. A component should propagate this field from an input buffer to its associated output buffer. The list of flags is as follows:

```
#define OMX_BUFFERFLAG_EOS 0x00000001
#define OMX_BUFFERFLAG_STARTTIME 0x00000002
#define OMX_BUFFERFLAG_DECODEONLY 0x00000004
#define OMX_BUFFERFLAG_DATACORRUPT 0x00000008
```

```

#define OMX_BUFFERFLAG_ENDOFFRAME 0x00000010
#define OMX_BUFFERFLAG_SYNCFRAME 0x00000020
#define OMX_BUFFERFLAG_EXTRADATA 0x00000040
#define OMX_BUFFERFLAG_CODECCONFIG 0x00000080
#define OMX_BUFFERFLAG_TIMESTAMPINVALID 0x00000100
#define OMX_BUFFERFLAG_READONLY 0x00000200
#define OMX_BUFFERFLAG_ENDOFSUBFRAME 0x00000400
#define OMX_BUFFERFLAG_SKIPFRAME 0x00000800

```

- OMX\_BUFFERFLAG\_EOS is set by a source component (e.g. a demuxer) sets OMX\_BUFFERFLAG\_EOS when it has reached the end of the stream content on a particular output port. Some examples of this are:
  - End of a stream within a 3GP file,
  - Camera Component stopping the emission of stream data on its capture port. i.e. OMX\_IndexAutoPauseAfterCapture support

The emission of the OMX\_BUFFERFLAG\_EOS does not preclude the possibility of subsequent stream content being emitted on the port in response to an IL client command. In the examples above, a port may emit additional stream content when:

- It receives a seek request to an earlier position earlier in the 3GP file,
- The Camera Component is requested to start emitting additional content via the capture port.

Other components forward the OMX\_BUFFERFLAG\_EOS in a way that is appropriate for their processing.

OMX\_BUFFERFLAG\_EOS shall not be emitted in response to a state change command.

- OMX\_BUFFERFLAG\_STARTTIME is set by the source of a stream (e.g., a de-multiplexing component) sets the OMX\_BUFFERFLAG\_STARTTIME flag on the buffer that contains the starting timestamp for the stream. The starting timestamp corresponds to the first data that should be displayed at startup or after a seek operation.

The first timestamp of the stream is not necessarily the start time. For instance, in the case of a seek to a particular video frame, the target frame may be an interframe. Thus the first buffer of the stream will be the intraframe preceding the target frame, and the start time will occur with the target frame along with any other required frames required to reconstruct the target intervening.

The OMX\_BUFFERFLAG\_STARTTIME flag is directly associated with the buffer timestamp. Thus, the association of the OMX\_BUFFERFLAG\_STARTTIME flag to buffer data and its propagation is identical to that of the timestamp.

A clock component client that receives a buffer with the OMX\_BUFFERFLAG\_STARTTIME flag shall perform an OMX\_SetConfig call on its sync port using OMX\_IndexConfigTimeClientStartTime and pass the timestamp for the buffer.

- OMX\_BUFFERFLAG\_DECODEONLY is set by the source of a stream (e.g., a de-multiplexing component) sets the **OMX\_BUFFERFLAG\_DECODEONLY** flag on any buffer that should be decoded but not rendered. This flag is used, for instance, when a source seeks to a target interframe that requires decoding of frames preceding the target to facilitate reconstruction of the target. In this case, the source would emit the frames preceding the target downstream but mark them as decode only.

The OMX\_BUFFERFLAG\_DECODEONLY flag is associated with buffer data and propagated in a manner identical to that of the buffer timestamp.

A component that renders data should ignore all buffers with the OMX\_BUFFERFLAG\_DECODEONLY flag set.

- OMX\_BUFFERFLAG\_DATACORRUPT flag is set when the client identifies the data in the associated buffer is identified as corrupt.
- OMX\_BUFFERFLAG\_ENDOFFRAME is an optional flag that is set by an output port when the last byte that a buffer payload contains is an end-of-frame. Any component that implements setting the OMX\_BUFFERFLAG\_ENDOFFRAME flag on an output port shall set this flag for every buffer sent from the output port containing an end-of-frame. No buffer payload can contain data from two separate frames.

These restrictions enable input ports that receive data from the output port to detect an end-of-frame without requiring additional processing. These restrictions also enable an input port to easily detect if an output port supports this flag by its presence or absence on completion of the first frame.

- The OMX\_BUFFERFLAG\_SYNCFRAME flag should be set by an output port to indicate that the buffer content contains a coded synchronization frame. A coded synchronization frame is a frame that can be reconstructed without reference to any other frame information. An example of a video synchronization frame is an MPEG4 I-VOP.

If the OMX\_BUFFERFLAG\_SYNCFRAME flag is set then the buffer may only contain one frame.

- The OMX\_BUFFERFLAG\_EXTRADATA is an optional flag that should be set by an output port when the buffer payload contains additional information appended to the end of the buffer payload. Each extra block of data is preceded by an OMX\_OTHER\_EXTRADATATYPE structure which

provides specific information about the extra data. The OMX\_BUFFERFLAG\_EXTRADATA is used to identify the availability of additional information after the buffer payload. Each extra block of data is preceded by an OMX\_OTHER\_EXTRADATATYPE structure, which provides specific information about the extra data.

Extra data shall only be present when the OMX\_BUFFERFLAG\_EXTRADATA is signaled.

- The OMX\_BUFFERFLAG\_CODECCONFIG is an optional flag that is set by an output port when all bytes in the buffer form part or all of a set of codec specific configuration data. Examples include SPS/PPS NAL units for OMX\_VIDEO\_CodingAVC or AudioSpecificConfig data for OMX\_AUDIO\_CodingAAC. Any component that for a given stream sets OMX\_BUFFERFLAG\_CODECCONFIG shall not mix codec configuration bytes with frame data in the same buffer, and shall send all buffers containing codec configuration bytes before any buffers containing frame data that those configurations bytes describe. If the stream format for a particular codec has a frame specific header at the start of each frame, for example OMX\_AUDIO\_CodingMP3 or OMX\_AUDIO\_CodingAAC in ADTS mode, then these shall be presented as normal without setting OMX\_BUFFERFLAG\_CODECCONFIG.
- OMX\_BUFFERFLAG\_TIMESTAMPINVALID is set to indicates that the nTimeStamp parameter does not contain valid timestamp information. A component emitting stream content with invalid or no timestamp information shall set this flag to indicate presence of no timestamp information. A component that updates the nTimeStamp parameter with valid timestamp information shall clear this flag.
- OMX\_BUFFERFLAG\_READONLY This flag is set when a component emitting the buffer on an output port or the IL client wishes to identify the buffer payload contents to be read-only. An IL client or an input port of a component shall not alter the contents of the buffer. This flag shall only be cleared by the originator of component emitting the buffer when the buffer is returned to it. For tunneled ports, the usage of this flag shall be allowed only if the components negotiated a read-only tunnel.
- OMX\_BUFFERFLAG\_ENDOFSUBFRAME is an optional flag set by an output port when the last byte of a buffer payload aligns with an end-of-sub-frame, where a sub-frame represents an independently decodable unit that may be a subset of a frame (e.g., a NAL unit in case of AVC, a slice in case of MPEG4 video, a field in case of VC-1 etc.). Any component that implements setting the OMX\_BUFFERFLAG\_ENDOFSUBFRAME flag

on an output port shall set this flag for every buffer sent from the output port if the last byte of the buffer payload aligns with an end-of-sub-frame.  
A component that implements setting the  
OMX\_BUFFERFLAG\_ENDOFSUBFRAME flag on an output port may  
apply this flag on buffers whose payloads contain data from more than one  
sub-frame.

If a component implements and sets both  
OMX\_BUFFERFLAG\_ENDOFFRAME flag and  
OMX\_BUFFERFLAG\_ENDOFSUBFRAME flag on an output port, it shall  
apply both of these flags on a buffer whose last byte of buffer payload  
aligns with an end-of-frame.  
These restrictions enable input ports that receive data from the output port  
to detect end-of-sub-frames without requiring additional processing.

- OMX\_BUFFERFLAG\_SKIPFRAME is set to indicate the presence of a skipped (not coded) frame within the stream that needs to be signaled out-of-band from the stream content. Each Skipped frame has to be signaled separately. These restrictions enable input ports to detect the presence of skipped frames without requiring additional processing.  
For example, codecs such as VC1 Simple and Main profile (without B-frames) do not support the ability to signal skipped frames in-band (within the video stream syntax), skipped frames are signaled externally (i.e. out-of-band) within the media containers.

- nOutputPortIndex contains the port index of the output port that uses the buffer. If a buffer header is used on an input port that is communicating with the IL client, the value of nOutputPortIndex is undefined.
- nInputPortIndex contains the port index of the input port that uses the buffer. If a buffer header is used on an input-output port that is communicating with the IL client, the value of nInputPortIndex is undefined.

### 3.1.2.93.1.3.8 OMX\_PORT\_PARAM\_TYPE

A component uses the OMX\_PORT\_PARAM\_TYPE structure to identify the number and starting index of ports of a particular domain.

OMX\_PORT\_PARAM\_TYPE is defined as follows.

```
typedef struct OMX_PORT_PARAM_TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPorts;
    OMX_U32 nStartPortNumber;
} OMX_PORT_PARAM_TYPE;
```

### **3.1.2.9.13.1.3.8.1 Parameter Definitions**

- nPorts is the number of ports of a given port domain (audio, video, image, or other) for the component.
- nStartPortNumber is the index of the first port of a given port domain (audio, video, image, or other) for the component . Subsequent ports of the given domain are numbered sequentially from nStartPortNumber.

### **3.1.2.103.1.3.9 OMX\_CALLBACKTYPE**

The OpenMAX IL includes a callback mechanism that allows a component to communicate ~~the following~~ with the IL client.<sup>2</sup>

- ~~An asynchronous command triggered by the IL client has completed successfully or failed and generated an error. Commands include those sent by OMX\_SendCommand and those implied by IL client calls to EmptyThisBuffer or FillThisBuffer.~~
- ~~An error unassociated with a command triggered by the IL client has occurred. For example, the component has lost resources it needs for processing, is suspended, and transitions suffered an unrecoverable error and is transitioning to the OMX\_StatePausedInvalid state.~~

To accomplish a callback, the OpenMAX IL has three callback functions defined: a generic event handler and two callbacks related to the dataflow (EmptyBufferDone and FillBufferDone).

The IL client is responsible for filling in an OMX\_CALLBACKTYPE structure with its callback entry points and passing the structure to the OpenMAX IL core at initialization (init) time, usually in the OMX\_GetHandle function.

OMX\_CALLBACKTYPE is defined as follows.

```
typedef struct OMX_CALLBACKTYPE
{
    OMX_ERRORTYPE (*EventHandler)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_PTR pAppData,
        OMX_IN OMX_EVENTTYPE eEvent,
        OMX_IN OMX_U32 nData1,
        OMX_IN OMX_U32 nData2,
        OMX_IN OMX_PTR pEventData);
    OMX_ERRORTYPE (*EmptyBufferDone)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_PTR pAppData,
        OMX_IN OMX_BUFFERHEADERTYPE* pBuffer);
    OMX_ERRORTYPE (*FillBufferDone)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_PTR pAppData,
        OMX_IN OMX_BUFFERHEADERTYPE* pBuffer);
} OMX_CALLBACKTYPE;
```

### 3.1.2.10.43.1.3.9.1 EventHandler

A component uses the EventHandler method to notify the IL client when an event of interest occurs within the component. The OMX\_EVENTTYPE enumeration defines the set of OpenMAX IL events; refer to the definition of this enumeration for the meaning of each event. ~~The information carried within nData1, carries the value of OMX\_COMMANDTYPE that has been completed or OMX\_ERRORTYPE. nData2 and carries further event parameters, e.g., OMX\_STATETYPE. pEventData varies depending on OMX\_EVENTTYPE, refer to Table 3-11 for specific details.~~  
~~contains event specific data. The pEventData pointer may contain additional data associated with the event (e.g., mark specific data). Note that in the case of OMX\_EventCmdComplete, the component shall cast the error code into OMX\_PTR for storing it into pEventData parameter. Upon receiving the OMX\_EventCmdComplete event, the IL client shall cast the pEventData parameter back into OMX\_ERRORTYPE.~~ A call to EventHandler is a blocking call, so the IL client should ~~respond return~~ within five milliseconds to avoid blocking the component for an excessively long time period.

The EventHandler method is defined as follows.

```
OMX_ERRORTYPE(* OMX_CALLBACKTYPE::EventHandler)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_PTR pAppData,  
    OMX_IN OMX_EVENTTYPE eEvent,  
    OMX_IN OMX_U32 nData1,  
    OMX_IN OMX_U32 nData2,  
    OMX_IN OMX_PTR pEventData)
```

The parameters are as follows.

**Parameter**

**Description**

hComponent      The handle of the component that calls this function.  
[in]

pAppData      A pointer to IL client-defined data.  
[in]

eEvent      The event that the component is communicating to the IL client.  
[in]

nData1      The first integer event-specific parameter. See Table 3-11 for the meaning in the context  
[in] of each event.

nData2      The second integer event-specific parameter. See Table 3-11 for the meaning in the context  
[in] of each event. The default value is 0 if not used.

pEventData      A pointer to additional event-specific data or an error code. See Table 3-11 for the  
[in] meaning in the context of each event.

Table 3-11 lists the parameters used in each event.

**Table 3-11: Event Parameter Usage**

eEvent	nData1	nData2	pEventData
OMX_EventCmdComplete	OMX_CommandStateSet	State reached	<u>Error codeNull</u>
	OMX_CommandFlush	Port index	<u>Error codeNull</u>
	OMX_CommandPort Disable	Port index	<u>Error codeNull</u>
	OMX_CommandPort Enable	Port index	<u>Error codeNull</u>
	OMX_CommandMark Buffer	Port index	<u>Error codeNull</u>
OMX_EventError	Error code	<u>In some cases, this field is used to convey additional information linked to the error event (see Table 3-12). ØZero when no additional information is needed.Ø</u>	Null
OMX_EventMark	0	0	Data linked to the mark, if any
OMX_EventPortSettings Changed	Port index	<u>Param or config indexØ</u>	Null
OMX_EventBufferFlag	Port index	nFlags unaltered	Null
OMX_EventResources Acquired	0	0	Null
<u>OMX_EventComponentRes umeđ</u>	<u>0</u>	<u>0</u>	<u>Null</u>
OMX_EventDynamic ResourcesAvailable	0	0	Null
<u>OMX_EventPortFormatDe tected</u>	<u>0</u>	<u>0</u>	<u>Null</u>
<u>OMX_EventIndexSetting Changed</u>	<u>Port index (may also be OMX_ALL)</u>	<u>Param or config index</u>	<u>Null</u>
<u>OMX_EventPortNeedsDis able</u>	<u>Port Index</u>	<u>Requesting Port Index</u>	<u>Null</u>
<u>OMX_EventPortNeedsFlu sh</u>	<u>Port Index</u>	<u>Requesting Port Index</u>	<u>Null</u>

The following table lists the error codes that use the nData2 field to carry additional information related to the error.

**Table 3.12: OMX\_EventError Event nData2 Parameter Usage**

<u>nData1 (error code)</u>	<u>nData2</u>
<u>OMX_ErrorUnderflow,</u> <u>OMX_ErrorOverflow,</u> <u>OMX_ErrorStreamCorrupt,</u> <u>OMX_ErrorStreamCorruptStalled,</u> <u>OMX_ErrorStreamCorruptFatal,</u> <u>OMX_ErrorPortUnpopulated,</u> <u>OMX_ErrorFormatNotDetected</u>	<u>The index of the affected port.</u>

### 3.1.2.10.23.1.3.9.2 EmptyBufferDone

A component uses the EmptyBufferDone callback to pass a buffer from an input port back to the IL client. A component sets updates the nOffset and nFilledLen values of the buffer header to reflect the portion of the buffer it consumed; for example, nFilledLen is set equal to 0x0 if completely consumed.

In addition to facilitating normal data flow between an executing component and the IL client, a component uses the EmptyBufferDone function to return input buffers to the IL client in the following cases:

- The IL client commands a transition from OMX\_StateExecuting or OMX\_StatePause to OMX\_StateIdle or to OMX\_StateInvalid.
- The IL client flushes or disables a port.

In these cases, A component may also return a partially consumed input buffer to the IL client in the following cases:

- The IL client commands a transition from OMX\_StateExecuting or OMX\_StatePause to OMX\_StateIdle.
- The IL client flushes or disables a port.

The EmptyBufferDone call is a blocking call that should return from within five milliseconds. Therefore, the IL client may elect not to fill the buffers during this call but queue them for processing outside this call.

The EmptyBufferDone call is defined as follows.

```
OMX_ERRORTYPE (* OMX_CALLBACKTYPE::EmptyBufferDone) (
    OMX_OUT_IN OMX_HANDLETYPE hComponent,
    OMX_OUT_IN OMX_PTR pAppData,
    OMX_OUT_IN OMX_BUFFERHEADERTYPE* pBuffer)
```

The parameters are as follows.

<b>Parameter</b>	<b>Description</b>
hComponent [ <i>out/in</i> ]	The handle of the component that is calling this function.
pAppData [ <i>out/in</i> ]	A pointer to IL client-defined data.
pBuffer [ <i>out/in</i> ]	A pointer to an OMX_BUFFERHEADERTYPE structure that was consumed or returned.

### 3.1.2.10.33.1.3.9.3 FillBufferDone

A component uses the `FillBufferDone` callback to pass a buffer from an output port back to the IL client. A component sets the `nOffset` and `nFilledLen` of the buffer header to reflect the portion of the buffer it filled; for example, `nFilledLen` is equal to `0x0` if it contains no data).

In addition to facilitating normal dataflow between an executing component and the IL client, a component uses this function to return output buffers to the IL client in the following cases:

- The IL client commands a transition from `OMX_StateExecuting` or `OMX_StatePause` to `OMX_StateIdle` or to `OMX_StateInvalid`.
- The IL client flushes or disables a port.

The `FillBufferDone` call is a blocking call that should return from within five milliseconds. The IL client may elect not to empty the buffers during this call but queue them for consumption outside this call.

`FillBufferDone` is defined as follows.

```
OMX_ERRORTYPE (* OMX_CALLBACKTYPE::FillBufferDone) (
    OMX_OUT_IN OMX_HANDLETYPE hComponent,
    OMX_OUT_IN OMX_PTR pAppData,
    OMX_OUT_IN OMX_BUFFERHEADERTYPE* pBuffer)
```

The parameters are as follows.

<b>Parameter</b>	<b>Description</b>
hComponent [ <i>out/in</i> ]	The handle of the component to access. This handle is the component handle returned by the call to the <code>GetHandle</code> function.
pAppData [ <i>out/in</i> ]	A pointer to IL client-defined data
pBuffer [ <i>out/in</i> ]	A pointer to an OMX_BUFFERHEADERTYPE structure that was filled or returned.

### 3.1.2.113.1.3.10 OMX\_PARAM\_BUFFERSUPPLIERTYPE

The `OMX_PARAM_BUFFERSUPPLIERTYPE` structure is used to communicate buffer supplier settings or buffer supplier preferences [during tunnel setup \(see Section 3.4.1.2\)](#).

OMX\_PARAM\_BUFFERSUPPLIERTYPE is defined as follows.

```
typedef struct OMX_PARAM_BUFFERSUPPLIERTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BUFFERSUPPLIERTYPE eBufferSupplier;
} OMX_PARAM_BUFFERSUPPLIERTYPE;
```

#### **3.1.2.11.13.1.3.10.1 Parameter Definitions**

- nPortIndex represents the port that this structure applies to.
- eBufferSupplier is a field that specifies the port in the tunnel that is the supplier port (see Section 3.1.1.5).  
contains the index of the buffer supplier, if input or output.

#### **3.1.2.123.1.3.11 OMX\_TUNNELSETUPTYPE**

The ComponentTunnelRequest function uses the OMX\_TUNNELSETUPTYPE structure to pass data between two ports when an IL client connects these ports via an OMX\_SetupTunnel call.

OMX\_TUNNELSETUPTYPE is defined as follows.

```
typedef struct OMX_TUNNELSETUPTYPE {
    OMX_U32 nTunnelFlags;
    OMX_BUFFERSUPPLIERTYPE eSupplier;
} OMX_TUNNELSETUPTYPE;
```

#### **3.1.2.12.13.1.3.11.1 Parameter Definitions**

- nTunnelFlags is an integer parameter that contains one or more bit flags applied to the port that receives this structure. Flags include:

```
#define OMX_PORTTUNNELFLAG_READONLY 0x00000001
```

If the flag is set as read only, the input port that receives this structure cannot alter the contents of buffers supplied on the tunnel.

- The eSupplier field defines whether the input port or the output port provides the buffers. The exact sequence of calls to set up a tunnel is specified in section 3.4.1.2.

#### **3.1.2.133.1.3.12 OMX\_PARAM\_PORTDEFINITIONTYPE**

The OMX\_PARAM\_PORTDEFINITIONTYPE structure contains a set of generic fields that characterize each port of the component. Some of these fields are common to all domains while other fields are specific to their respective domains. The IL client uses this structure to retrieve general information from each port.

OMX\_PARAM\_PORTDEFINITIONTYPE is defined as follows.

```

typedef struct OMX_PARAM_PORTDEFINITIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_DIRTYPE eDir;
    OMX_U32 nBufferCountActual;
    OMX_U32 nBufferCountMin;
    OMX_U32 nBufferSize;
    OMX_BOOL bEnabled;
    OMX_BOOL bPopulated;
    OMX_PORTDOMAINTYPE eDomain;
    union {
        OMX_AUDIO_PORTDEFINITIONTYPE audio;
        OMX_VIDEO_PORTDEFINITIONTYPE video;
        OMX_IMAGE_PORTDEFINITIONTYPE image;
        OMX_OTHER_PORTDEFINITIONTYPE other;
    } format;
    OMX_BOOL bBuffersContiguous;
    OMX_U32 nBufferAlignment;
} OMX_PARAM_PORTDEFINITIONTYPE;

```

### 3.1.2.13.13.1.3.12.1 Parameter Definitions

- nPortIndex represents the port that this structure applies to is a read-only field the identifies the port. The value of nPortIndex is a unique 32-bit number for the component. No two ports on a single component may share the same port number, but ports on different components may have the same port number.
- eDir is a read-only field that indicates the direction (OMX\_DirInput or OMX\_DirOutput) for the port.

Table 3-13: OMX\_DIRTYPE enumeration

Field Name	Description
OMX_DirInput	Input port
OMX_DirOutput	Output port

- nBufferCountActual represents the number of buffers that are required on this port before it is populated, as indicated by the bPopulated field of this structure. The component shall set a default value equal to no less than nBufferCountMin for this field. The component shall disallow changes to this field when requested to set a value less than nBufferCountMin.
- nBufferCountMin is a read-only field that specifies the minimum number of buffers that the port requires. The component shall define this non-zero default value.
- nBufferSize is a read-only field that specifies the minimum size in bytes for buffers that are allocated for this port. .
- bEnabled is a read-only Boolean field that indicates if the port is enabled. Ports default to bEnabled = OMX\_TRUE and are enabled/disabled by sending the

OMX\_CommandPortEnable and OMX\_CommandPortDisable commands with the OMX\_SendCommand method. A port shall not be populated when it is not enabled.

- bPopulated is a read-only Boolean field that indicates if a port is populated. A port is populated when all of the buffers indicated by nBufferCountActual with a size of at least nBufferSize have been allocated on the port. A populated port shall be enabled. Enabled ports shall be populated on a transition to OMX\_StateIdle and unpopulated on a transition to OMX\_StateLoaded.
- eDomain is a read-only field that indicates the domain of the port. This field determines the contents of the format union explained in the next paragraph.  
Values for eDomain are defined in Table 3-14 below.

**Table 3-14: Port Domains**

Field Name	Description
<a href="#">OMX_PortDomainAudio</a>	Specifies that the field format is a structure of the OMX_AUDIO_PORTDEFINITIONTYPE type.
<a href="#">OMX_PortDomainVideo</a>	Specifies that the field format is a structure of the OMX_VIDEO_PORTDEFINITIONTYPE type.
<a href="#">OMX_PortDomainImage</a>	Specifies that the field format is a structure of the OMX_IMAGE_PORTDEFINITIONTYPE type.
<a href="#">OMX_PortDomainOther</a>	Specifies that the field format is a structure of the OMX_OTHER_PORTDEFINITIONTYPE type.

- The format fields are a union of domain-specific parameters. For more information on parameters for audio, video, image, and other domains, see Section 4 - OpenMAX IL Data API.
- bBuffersContiguous is a read only Boolean field that has two meanings depending on the component state and whether the port is enabled or disabled. When the port is disabled or the component is in the state OMX\_StateLoaded, bBuffersContiguous shall indicate whether the this port prefers requires each buffer to be in physically contiguous memory. During the transition from OMX\_StateLoaded to OMX\_StateIdle, or when enabling the port, the buffer allocator shall update this field to indicate the actual contiguity of the buffers as described in section 2.8.2.  
Note that having physically contiguous buffers is not always enough to allow processing by hardware, the exact meaning of this field may be platform specific.
- nBufferAlignment is a read-only field that specifies the alignment the port requires for each of its buffers (OMX\_BUFFERHEADERTYPE structure's pBuffer parameter) and the payload data within the buffer (OMX\_BUFFERHEADERTYPE structure's pBuffer + nOffset parameters).

For example, (e.g.—a value of 4 denotes the alignment at each buffer shall be 4-byte aligned). A value of zero denotes that the port does not have any alignment restrictions. Buffer alignments are restricted to powers of two.

### **3.1.3.13 OMX\_CONFIG\_TUNNELEDPORTSTATUSTYPE**

The OMX\_CONFIG\_TUNNELEDPORTSTATUSTYPE structure is used to communicate to a port what API calls are allowed on its tunneled port.

This structure is intended to be used in interop profile only. A component shall use this structure to inform its tunneled components about when some specific API calls become allowed on its non-supplier ports.

The index specified for this structure is OMX\_IndexConfigTunneledPortStatus. OMX\_CONFIG\_TUNNELEDPORTSTATUSTYPE is defined as follows:

```
typedef struct OMX_CONFIG_TUNNELEDPORTSTATUSTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nTunneledPortStatus;
} OMX_CONFIG_TUNNELEDPORTSTATUSTYPE;
```

#### **3.1.3.13.1 Parameters**

- nPortIndex represents the port that this structure applies to.
- nTunneledPortStatus is an integer parameter that contains one or more bit flags applied to the port that receives this structure.

**Table 3-15 : Port Status Flags**

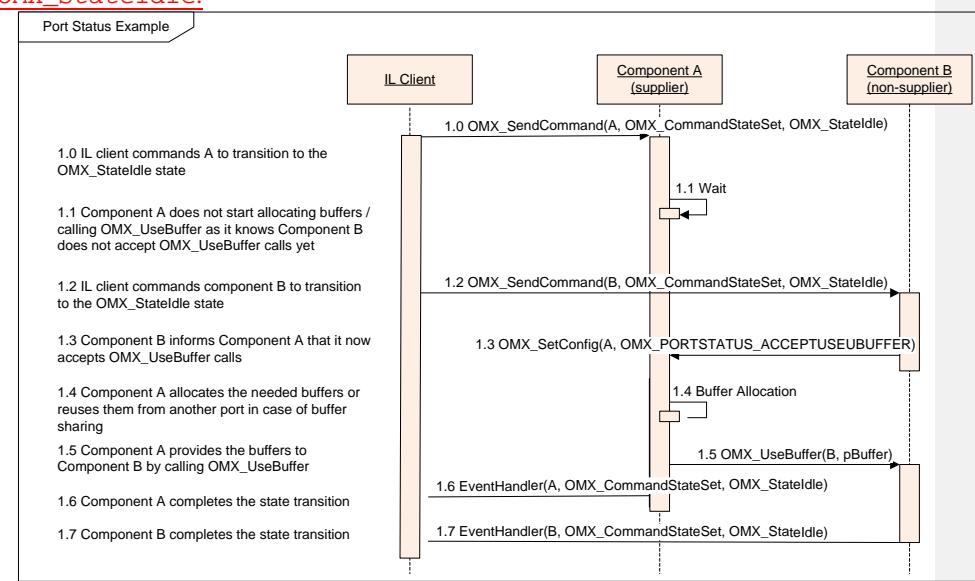
<u>Flags</u>	<u>Description</u>
<u>OMX_PORTSTATUS_ACCEPTUSEBUFFER</u>	When this flag is set for a port, it means that its tunneled port accepts <u>OMX_UseBuffer</u> calls
<u>OMX_PORTSTATUS_ACCEPTBUFFEREXCHANGE</u>	When this flag is set for a port, it means that its tunneled port would accept <u>OMX_EmptyThisBuffer</u> or <u>OMX_FillThisBuffer</u> calls.

#### **3.1.3.13.2 OMX\_StateLoaded to OMX\_StateIdle Transition**

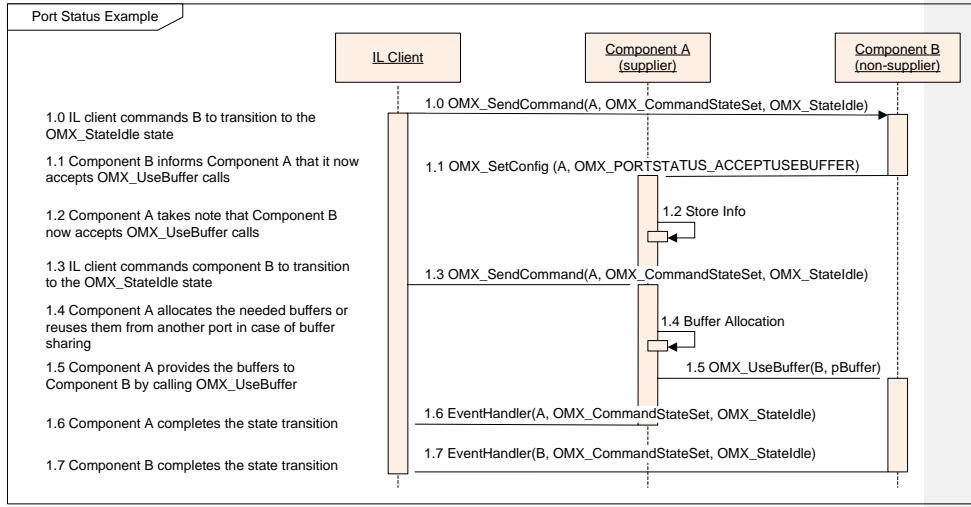
When commanded to transition from OMX\_StateLoaded state to OMX\_StateIdle state, a component does the following:

- For each enabled non-supplier port, it shall inform its tunneled port using `OMX_SetConfig(OMX_IndexConfigTunneledPortStatus)` that `OMX_UseBuffer` calls are now allowed.

For each enabled supplier port, it should start calling `OMX_UseBuffer` only when it got the information that the call would be accepted by the tunneled port. Attempting to call `OMX_UseBuffer` prior to getting the `OMX_SetConfig(OMX_IndexConfigTunneledPortStatus)` telling that the call would be accepted is likely to result in the call returning `OMX_ErrorIncorrectStateOperation`. If one of the `UseBuffer` calls returns `OMX_ErrorIncorrectStateOperation`, the component should wait for a call to `OMX_SetConfig(OMX_PORTSTATUS_ACCEPTUSEBUFFER)` to resume the sequence, or for a `OMX_SendCommand(CommandStateSet, OMX_StateLoaded)` that cancels the current transition to `OMX_StateIdle`.



**Figure 3-2 : OMX\_StateLoaded to OMX\_StateIdle transition while supplier first**

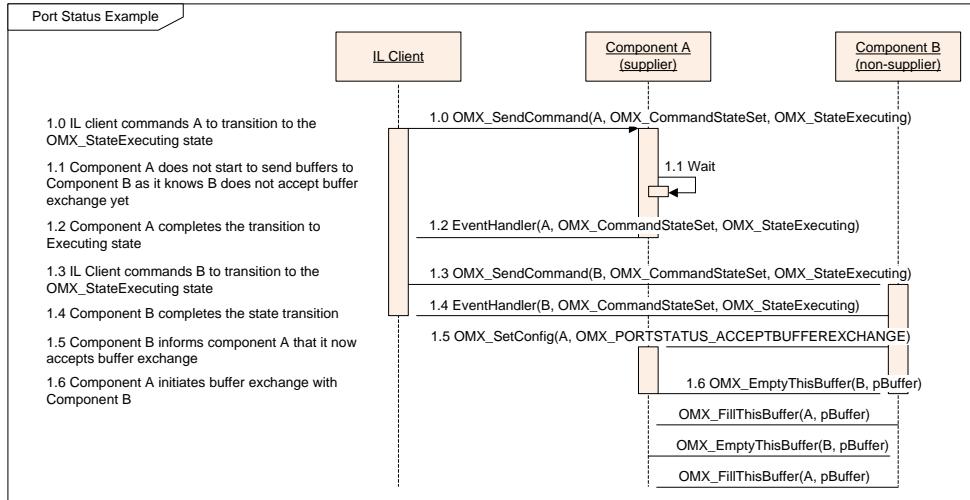


**Figure 3-3: OMX\_StateLoaded to OMX\_StateIdle transition while non-supplier first**

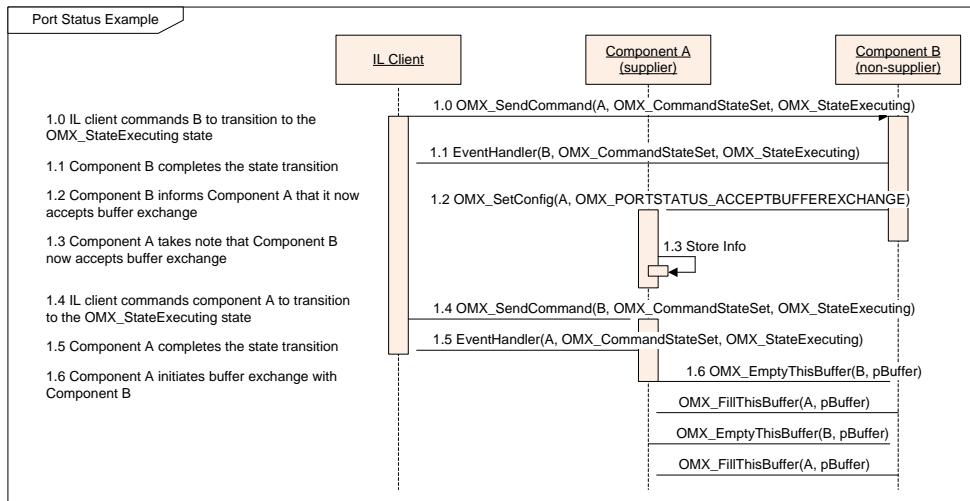
### 3.1.3.13.3 OMX\_StateIdle to OMX\_StateExecuting Transition

When commanded to transition from OMX\_StateIdle state to OMX\_StateExecuting state, a component does the following:

- For each enabled non-supplier port, it shall inform its tunneled port using `OMX_SetConfig(OMX_IndexConfigTunneledPortStatus)` that buffer exchange is now allowed.
- For each enabled supplier port, it should start calling `OMX_EmptyThisBuffer` / `OMX_FillThisBuffer` only when it got the information that the call would be accepted by the tunneled port. Attempting to call `OMX_EmptyThisBuffer` or `OMX_FillThisBuffer` prior to getting the `OMX_SetConfig(OMX_IndexConfigTunneledPortStatus)` telling that the call would be accepted is likely to result in the call returning `OMX_ErrorIncorrectStateOperation`.



**Figure 3-4: OMX StateIdle to OMX StateExecuting transition while supplier first**



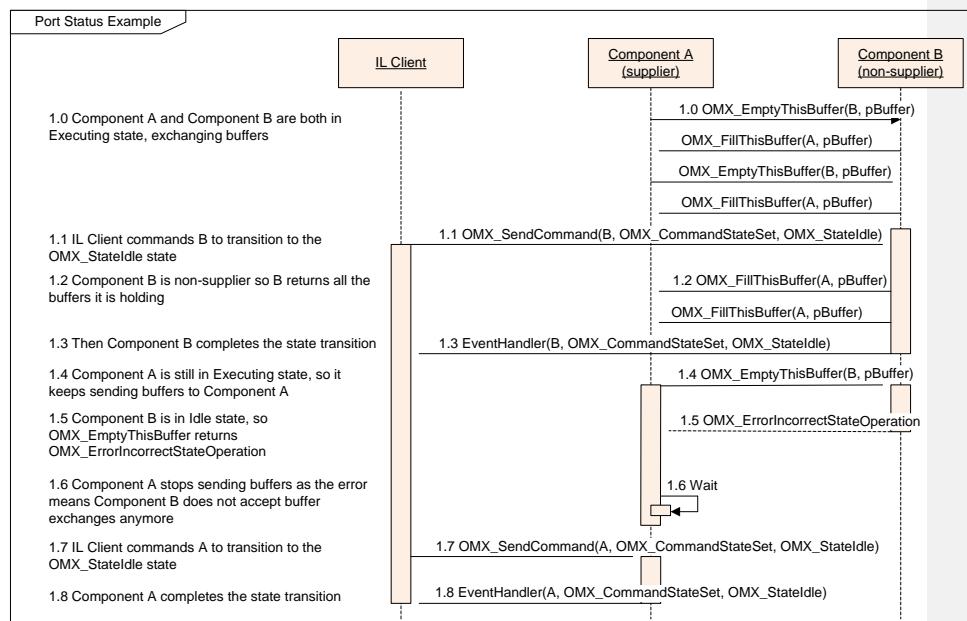
**Figure 3-5: OMX StateIdle to OMX StateExecuting State Transition while non-supplier first**

### 3.1.3.13.4 OMX StateExecuting to OMX StateIdle Transition

When commanded to transition from OMX\_StateExecuting state to OMX\_StateIdle state, a component does the following:

- For each enabled non-supplier port, it shall return all the buffers that the port is holding using `OMX_EmptyThisBuffer` for an output port or `OMX_FillThisBuffer` for an input port, then it shall reject any further call to `OMX_EmptyThisBuffer` for an input port or `OMX_FillThisBuffer` for an output port by returning `OMX_ErrorIncorrectStateOperation`.
- For each enabled supplier port, it shall wait for all buffers to be returned by the tunneled port before completing the transition.

A component in `OMX_StateExecuting` state may get an `OMX_ErrorIncorrectStateOperation` when calling `OMX_EmptyThisBuffer` or `OMX_FillThisBuffer` if the non-supplier component is commanded to transition to `OMX_StateIdle` before the supplier component. In this situation, it should stop buffer exchanges until it commanded to transition to `OMX_State_Idle` or until it gets the `OMX_SetConfig(OMX_IndexConfigTunneledPortStatus)` telling that buffer exchanges are accepted again by the tunneled port (See Figure 3-5: `OMX_StateIdle` to `OMX_StateExecuting` State Transition while non-supplier firstFigure 3-12: `OMX_StateIdle` to `OMX_StateExecuting` State Transition while non-supplier first).



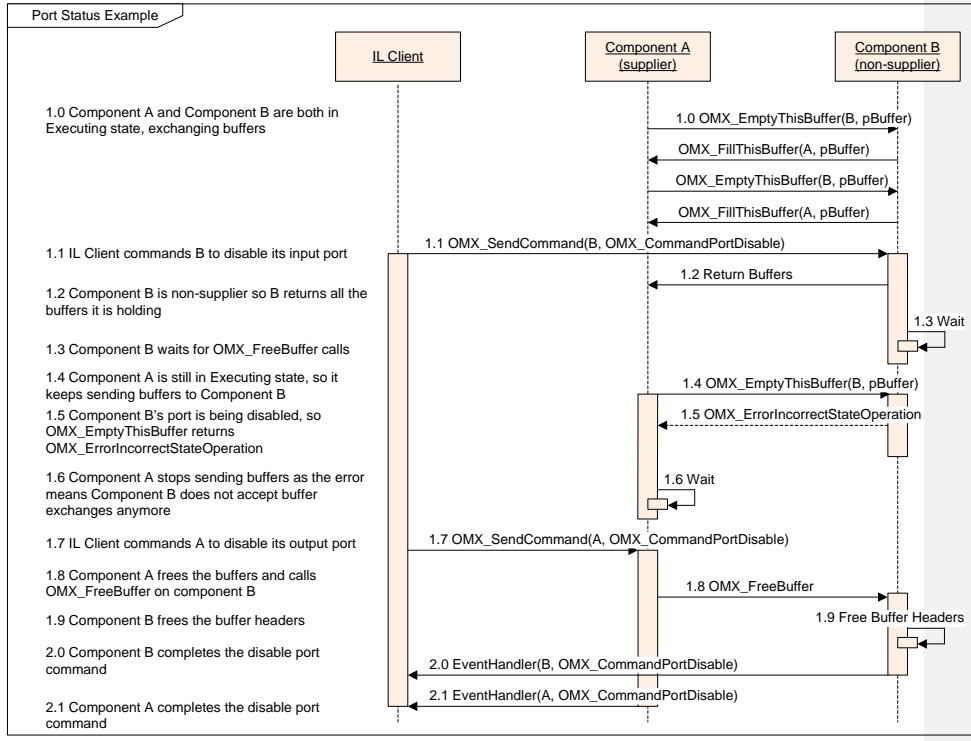
**Figure 3-6: Executing to `OMX_StateIdle` transition while non-supplier first**

### 3.1.3.13.5 Disabling Ports – OMX\_CommandPortDisable

When commanded to disable a port, a component does the following:

- For a non-supplier port, it shall return all the buffers that the port is holding using OMX\_EmptyThisBuffer for an output port or OMX\_FillThisBuffer for an input port, then it shall reject any further call to OMX\_EmptyThisBuffer for an input port or OMX\_FillThisBuffer for an output port by returning OMX\_ErrorIncorrectStateOperation. It shall then wait for OMX\_FreeBuffer calls to complete the command.
- For a supplier port, it shall wait for all buffers to be returned by the tunneled port, then call OMX\_FreeBuffer to free all the buffers.

A component in Executing state may get an OMX\_ErrorIncorrectStateOperation when calling OMX\_EmptyThisBuffer or OMX\_FillThisBuffer if the non-supplier port gets the OMX\_CommandPortDisable command before the supplier port. In this situation, it should stop buffer exchanges until it gets the OMX\_CommandPortDisable command or until it gets the OMX\_SetConfig(OMX\_IndexConfigTunneledPortStatus) telling that buffer exchanges are accepted again by the tunneled port (See Figure 3-5: OMX\_StateIdle to OMX\_StateExecuting State Transition while non-supplier first(Figure 3-12: OMX\_StateIdle to OMX\_StateExecuting State Transition while non-supplier first)).



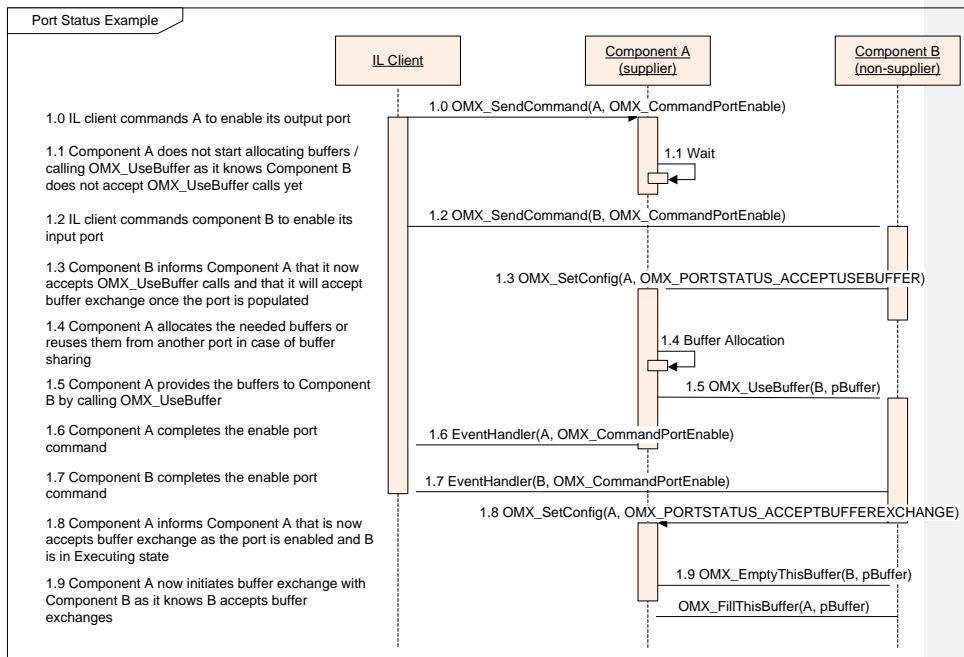
**Figure 3-7: Disabling ports while non-supplier first**

### 3.1.3.13.6 Enabling Ports – OMX\_CommandPortEnable

When commanded to enable a port, a component does the following:

- For a non-supplier port, it shall inform its tunneled port using `OMX_SetConfig(OMX_IndexConfigTunneledPortStatus)` that `OMX_UseBuffer` calls are now allowed. If the component is in `OMX_StateExecuting` state, it can inform in the same call that buffer exchange will be allowed once the port is populated.
- For a supplier port, it should start calling `OMX_UseBuffer` only when it got the information that the call would be accepted by the tunneled port. Attempting to call `OMX_UseBuffer` prior to getting the `OMX_SetConfig(OMX_IndexConfigTunneledPortStatus)` telling that the call would be accepted is likely to result in the call returning `OMX_ErrorIncorrectStateOperation`. If one of the `OMX_UseBuffer` calls returns `OMX_ErrorIncorrectStateOperation`, the component should wait for a call to `OMX_SetConfig(OMX_PORTSTATUS_ACCEPTUSEBUFFER)`.

to resume the sequence, or for a `OMX_SendCommand(OMX_CommandPortDisable)` that cancels the current transition to Enabled.



**Figure 3-8: enabling ports while supplier first**

### 3.1.3 `OMX_PORTDOMAINTYPE`

Table 3-12 enumerates the fields used in the `OMX_PARAM_PORTDEFINITIONTYPE` structure to define the domain of the port.

**Table 3-12: Port Domain Names**

Field Name	Description
<code>OMX_PortDomainAudio</code>	Specifies that the field format is a structure of the <code>OMX_AUDIO_PORTDEFINITIONTYPE</code> type.
<code>OMX_PortDomainVideo</code>	Specifies that the field format is a structure of the <code>OMX_VIDEO_PORTDEFINITIONTYPE</code> type.
<code>OMX_PortDomainImage</code>	Specifies that the field format is a structure of the <code>OMX_IMAGE_PORTDEFINITIONTYPE</code> type.

Field Name	Description
OMX_PortDomainOther	Specifies that the field format is a structure of the OMX_OTHER_PORTDEFINITIONTYPE type.

### **3.1.4 OMX\_HANDLETYPE**

The OMX\_HANDLETYPE structure defines the component handle as seen by the IL client. The component handle is used by the IL core to access all of the public methods of the component. The component handle also contains pointers to the private data area of the component. The OpenMAX IL core allocates and initializes the component handle with help from the component during the process of loading the component. After the component is successfully loaded, the IL client can safely access any of the public functions of the component via the IL Core macros, although some may return an error because the state of the component is inappropriate for the access. The component handle shall not be used directly by the IL Client or other IL components to access the public methods of an implementation. The IL Core macros shall be used instead.

## **3.2 OpenMAX IL Core Methods/Macros**

The OpenMAX IL core implements the main interface for an IL client that wants to use OpenMAX IL components. For efficiency, OpenMAX IL defines a set of OpenMAX IL core macros that map on one-to-one basis to most OpenMAX IL component methods.

Some macros and methods recommend that the function return within either five milliseconds or 20 milliseconds, depending on the function. The 5-millisecond timeout was deemed by the standards body to be a reasonable response time for commands that may not require buffer processing. The standards body identified the 20-millisecond timeout to be a reasonable response time for commands that may require buffer processing to be completed; the assumption here is that the longest buffer processing would be less than 30 milliseconds, which corresponds to 30-frames per second video. These timeouts are intended primarily to enable component integrators to get a good idea of component response latency via conformance testing.

The macros include the following:

- Get component information (version, capabilities).
- Set the callbacks structure from the IL client to the IL component.
- Set/Get component parameters at init time.
- Set/Get component parameters at run time.
- Allocate/De-allocate buffers.
- Send a buffer full of data to an OpenMAX IL component port.
- Send an empty buffer to an OpenMAX IL component port.
- Send commands to a component.

- Get the actual state of the component.
- Get references to OpenMAX IL component-proprietary parameters.

The OpenMAX IL core also implements methods for the following:

- Initializing/de-initializing the whole OpenMAX IL core.
- Getting an OpenMAX IL component handle.
- Releasing an OpenMAX IL component handle.
- Detecting all OpenMAX IL components available on the platform at run time.
- Setting up data tunnels among OpenMAX IL components.
- Tearing down data tunnels among OpenMAX IL components.
- Acquiring content pipes.
- Querying for information on installed standard component implementations.

When a time limit for the execution of a method is specified, it is not intended as a hard restriction for the conformance of the component to the standard, but if the limit is not respected, a note shall appear in the description document related to the component.

### **3.2.1    *Return Codes for the Functions***

Table 3-16 lists all of the possible return error codes for each function. A critical error denotes an error from which the component cannot recover. The component shall remain in its current state and wait for the action of the IL clients. If a client transitions to the OMX\_StateInvalid state when a critical error occurs after receiving a critical error event, the IL client shall transition the component from its current state back to OMX\_StateLoaded (through OMX\_StateIdle, where necessary), and unload it by calling OMX\_FreeHandle. Note that there is no guarantee that actions other than the necessary commands to transition the component to OMX\_StateLoaded state and unload it will succeed after the component has signaled the critical error. Despite the critical error event, the component shall return all buffers to their suppliers where necessary for state transition, and shall complete the OMX\_FreeBuffer calls.

All columns but the last two correspond to errors returned from a call to the component. The rightmost two columns denote errors sent asynchronously as the result of an internal error.

**Table 3-16: Error Codes**

	OMX_GetComponentVersion	OMX_SendCommand	OMX_SetParameter	OMX_SetConfig	OMX_SetExtensionIndex	OMX_GetState	OMX_AllocateBuffer	OMX_FreeBuffer	OMX_UsedImage	OMX_IsThisBuffer	OMX_EmptyThisBuffer	OMX_FillThisBuffer	OMX_ComponentDerrit	OMX_Init	OMX_Deinit	OMX_ComponentNameEnum	OMX_GetHandle	OMX_FreeHandle	OMX_SetupTunnel	OMX_TearDownTunnel	OMX_GetComponentsOfRole	OMX_GetRoleOfComponent	OMX_GetCoreInterface	OMX_FreeCoreInterface	Sent with EventHandler	Critical error
OMX_ErrorNone	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
OMX_ErrorInsufficientResources		X					X	X	X						X	X									X	
OMX_ErrorUndefined	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
OMX_ErrorInvalidComponentName																									X	
OMX_ErrorComponentNotFound																			X						X	
OMX_ErrorInvalidComponent	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
OMX_ErrorBadParameter	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
OMX_ErrorNotImplemented																			X	X	X	X	X	X		
OMX_ErrorUnderflow																									X	
OMX_ErrorOverflow																									X	
OMX_ErrorHardware																									X	
OMX_ErrorInvalidState	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
OMX_ErrorStreamCorrupt																									X	
OMX_ErrorPortsNotCompatible																			X							
OMX_ErrorResourcesLost																									X	
OMX_ErrorNoMore		X	X																X			X	X			
OMX_ErrorVersionMismatch	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
OMX_ErrorNotReady		X	X	X	X																					
OMX_ErrorTimeout	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
OMX_ErrorSameState																									X	
OMX_ErrorResourcesPreeempted																									X	
OMX_ErrorPortUnresponsiveDuringAllocation																									*	
OMX_ErrorPortUnresponsiveDuringDeallocation																									*	
OMX_ErrorPortUnresponsiveDuringStop																									*	
OMX_ErrorIncorrectStateTransition																									X	

	<code>OMX_GetComponentVersion</code>	<code>OMX_SendCommand</code>	<code>OMX_SetParameter</code>	<code>OMX_SetParameter</code>	<code>OMX_SetConfig</code>	<code>OMX_SetConfig</code>	<code>OMX_GetExtensionIndex</code>	<code>OMX_GetState</code>	<code>OMX_useBuffer</code>	<code>OMX_AllocateBuffer</code>	<code>OMX_FreeBuffer</code>	<code>OMX_EmptyThisBuffer</code>	<code>OMX_FillThisBuffer</code>	<code>OMX_ComponentDeInit</code>	<code>OMX_Init</code>	<code>OMX_Deinit</code>	<code>OMX_ComponentNameEnum</code>	<code>OMX_GetHandle</code>	<code>OMX_FreeHandle</code>	<code>OMX_SetupTunnel</code>	<code>OMX_TearDownTunnel</code>	<code>OMX_GetComponentRole</code>	<code>OMX_SetRoleOfComponent</code>	<code>OMX_GetCoreInterface</code>	<code>OMX_FreeCoreInterface</code>	
<code>OMX_ErrorIncorrectStateOperation</code>		X							X X	X		X X X													Critical error	
<code>OMX_ErrorUnsupportedSetting</code>			X	X																						
<code>OMX_ErrorUnsupportedIndex</code>			X X X X X																							
<code>OMX_ErrorBadPortIndex</code>	X X X X X X								X *	X	X	X X X								X X						
<code>OMX_ErrorPortUnpopulated</code>																									X	
<code>OMX_errorComponentSuspended</code>																										
<code>OMX_ErrorDynamicResourcesUnavailable</code>																									X	
<code>OMX_ErrorMbErrorsInFrame</code>																									X	
<code>OMX_ErrorFormatNotDetected</code>																									X	
<code>OMX_ErrorSeparateTablesUsed</code>		X																								
<code>OMX_ErrorTunnelingUnsupported</code>																				X						
<code>OMX_ErrorInvalidMode</code>				X																						
<code>OMX_ErrorStreamCorruptStalled</code>																					X					
<code>OMX_ErrorStreamCorruptFatal</code>																					X					
<code>OMX_ErrorPortsNotConnected</code>																				X						
<code>OMX_ErrorTunnelingUnsupported</code>																				X						
<code>OMX_ErrorContentURINotSpecified</code>																					X					
<code>OMX_ErrorContentURIError</code>		X	X																		X					
<code>OMX_ErrorCommandCancelled</code>																					X					

### 3.2.2 Macros

This section describes the OpenMAX IL core macros. Note that some of these calls occur when only the caller is in the appropriate state to make the call (e.g. when tunneling) or when the component is transitioning from one state to another.

[Table 3-17](#) defines which macros may be called on a component in each component state.

**Table 3-17: Valid Component Calls**

	OMX_GetComponentVersion	OMX_SendCommand	OMX_GetParameter	OMX_SetParameter	OMX_GetConfig	OMX_SetConfig	OMX_GetExtensionIndex	OMX_GetState	OMX_UseBuffer	OMX_AllocateBuffer	OMX_FreeBuffer	OMX_EmptyThisBuffer	OMX_FillThisBuffer	OMX_SetupTunnel	OMX_TeardownTunnel	OMX_UseEGLImage	OMX_SetCallbacks	OMX_FreeHandle
OMX_StateLoaded	X X <sup>4</sup>	X X	X X	X X	X X	X X	X <sup>1</sup>	X <sup>1</sup>	X	X	X	X	X	X	X	X	X	
OMX_StateIdle	X X <sup>4</sup>	X		X X	X X	X X			X	X <sup>3</sup>	X <sup>3</sup>	X						
OMX_StateExecuting	X X	X		X X	X X	X X			X	X <sup>3</sup>	X <sup>3</sup>	X						
OMX_StatePause	X X	X		X X	X X	X X			X	X X	X X	X X						
OMX_StateWaitForResources	X X <sup>4</sup>	X X		X X	X X	X X	X X	X X	X			X				X		
OMX_StateInvalid								X	X	X	X			X	-	X		
Disabled Port	X X	X X	X X	X X	X X	X X	X X	X <sup>2</sup>	X <sup>2</sup>	X	X *	X *	X	X	X	X		

<sup>1</sup>: valid if already started transition to OMX\_StateIdle. In tunneled case for a non-supplier port, the component calls OMX\_SetConfig(OMX\_PORTSTATUS\_ACCEPTUSEBUFFER) on its tunneled component to indicate when it can start calling UseBuffer.

<sup>2</sup>: valid if already started transition to Enabled Port. In tunneled case for a non-supplier port, the component calls OMX\_SetConfig(OMX\_PORTSTATUS\_ACCEPTUSEBUFFER) on its tunneled component to indicate when it can start calling UseBuffer.

<sup>3</sup>: in tunneled case for a non-supplier port, the component calls OMX\_SetConfig(OMX\_PORTSTATUS\_ACCEPTBUFFEXCHANGE) on its tunneled component to indicate when it can start exchanging buffers.

<sup>4</sup>: not valid for OMX\_CommandMarkBuffer

### 3.2.2.1 OMX\_GetComponentVersion

The GetComponentVersion macro will query the component and returns information about it. This is a blocking call. The component should return from this call within five milliseconds.

The macro is defined as follows.

```
#define OMX_GetComponentVersion (
    hComponent,
    pComponentName,
    pComponentVersion,
    pSpecVersion,
    pComponentUUID)
    ((OMX_COMPONENTTYPE*)hComponent)->GetComponentVersion( \
```

```

    hComponent,                                \
    pComponentName,                            \
    pComponentVersion,                         \
    pSpecVersion,                             \
    pComponentUUID)                           \

```

The parameters are as follows.

Parameter	Description
<i>hComponent</i> [in]	The handle of the component that executes the command.
<i>pComponentName</i> [out]	A pointer to a component name string. Component names are strings limited to a length up to 127 bytes plus the trailing null for a maximum length of 128 bytes. An example of a valid component name is "OMX.<vendor_name>.AUDIO.DSP.MIXER\0". Names are assigned by the vendor, but shall start with "OMX." concatenated to the vendor specified string.
<i>pComponentVersion</i> [out]	A pointer to an OpenMAX IL version structure that the component will populate. The component will fill in a value that indicates the component version. Note that the component version is not the same as the OpenMAX IL specification version, which is found in all structures. The vendor of the component defines the component version and establishes its value.
<i>pSpecVersion</i> [out]	A pointer to an OpenMAX IL version structure that the component will populate. <i>pSpecVersion</i> is the version of the specification that the component was built against. <i>Note that this value may or may not match the version of the structure. For example, if the component was built against the version 2.0 specification but the IL client, which creates the structure, was built against the version 1.0 specification, the versions would be different.</i>
<i>pComponentUUID</i> [out]	A pointer to an UUID identifier that uniquely identifies the component. A component shall not be required to provide this information, it is optional information that a component may choose to provide.

### 3.2.2.1.1 Prerequisites for This Method

This method has no prerequisites.

### 3.2.2.1.2 Sample Code Showing Calling Sequence

The following sample code shows a calling sequence.

```

/* detect mismatch between IL client's and component's spec version */
OMX_GetComponentVersion(
    hComp,
    &CompName,
    &CompVersion,
    &CompSpecVersion);
if (CompSpecVersion != IlClientVersion){
    printf("ERROR: version mismatch\n");
}

```

### 3.2.2.2 OMX\_SendCommand

The OMX\_SendCommand macro will invoke a command on the component. This is a non-blocking call that should, at a minimum, validate command parameters but return within five milliseconds. ~~The component normally executes the command outside the context of the call, though a solution without threading may elect to execute it in context.~~ In either case, the component uses an event callback to notify the IL client of the results of the command once completed. If the component executes the command successfully, the component generates an OMX\_EventCmdComplete callback ~~with error code set to OMX\_ErrorNone~~. If the component fails to execute the command, the component generates an OMX\_EventCmdCompleteError callback and passes the appropriate error as a parameter.

Calling OMX\_SendCommand when there is already an on-going command being processed is allowed only in the following cases:

- When currently doing a state transition from OMX\_StateLoaded to OMX\_StateIdle, the IL client can try to cancel the transition by commanding the component to go back to OMX\_StateLoaded state. Depending on timings, the on-going command will either be canceled, or it may have time to complete normally before the cancellation is received. When canceled, the on-going command will report an OMX\_EventCmdComplete event with OMX\_ErrorCommandCanceled error code.
- When currently enabling a port (OMX\_CommandPortEnable), the IL client can try to cancel the operation by commanding the component to disable (OMX\_CommandPortDisable) the port. Depending on timings, the on-going command will either be canceled, or it may have time to complete normally before the cancellation is received. When canceled, the on-going command will report an OMX\_EventCmdComplete event with OMX\_ErrorCommandCanceled error code.
- When currently doing a state transition from OMX\_StateLoaded to OMX\_StateIdle, the IL client can command the component to disable (OMX\_CommandPortDisable) a port. This may allow the component to complete the transition to OMX\_StateIdle if the port was the one blocking the transition. Depending on timings, the commands will either complete in the order they were sent (case where the on-going transition to OMX\_StateIdle completes normally before the disable port command is received), or in the reverse order (case where the disable port command - OMX\_CommandPortDisable -unblocks the transition to OMX\_StateIdle).
- When currently disabling (OMX\_CommandPortDisable) a port and if the port is doing buffer sharing, the component may report an event OMX\_EventPortNeedsDisable to indicate to the IL client that it shall disable another port of the component to allow the current disable port command (OMX\_CommandPortDisable) to complete. In this situation, the IL client

shall call OMX\_CommandPortDisable to disable the requested port. The commands will complete in the reverse order they were sent.

- When currently flushing (OMX\_CommandFlush) a port and if the port is doing buffer sharing, the component may report an OMX\_EventPortNeedsFlush event to indicate to the IL client that it shall flush another port of the component to allow the current flush command to complete. In this situation, the IL client shall call OMX\_CommandFlush to flush the requested port. The commands will complete in the reverse order they were sent.
- Queuing several OMX\_CommandMarkBuffer commands is allowed. The number of OMX\_CommandMarkBuffer commands that can be queued is implementation specific.

In any other case, OMX\_SendCommand will return OMX\_ErrorIncorrectStateOperation if the IL client attempts to call it when there is already an on-going command being processed.

The component may elect to queue commands for later execution. The only restriction is that the completion shall be done in the same order as the requests arrived.

The macro is defined as follows.

```
#define OMX_SendCommand ( \
    hComponent, \
    Cmd, \
    nParam, \
    pCmdData) \
    ((OMX_COMPONENTTYPE*)hComponent)->SendCommand( \
        hComponent, \
        Cmd, \
        nParam, \
        pCmdData)
```

The parameters are as follows.

Parameter	Description
<i>hComponent</i>	The handle of the component that executes the command [in]
<i>Cmd</i>	Command for the component to execute [in]
<i>nParam</i>	Integer parameter for the command that is to be executed [in]
<i>pCmdData</i>	A pointer that contains implementation-specific data that cannot be represented with the numeric parameter <i>nParam</i> [in]

Section 3.3—[OpenMAX IL Component Methods and Structures](#)[OpenMAX IL Component Methods and Structures](#)[OpenMAX IL Component Methods and Structures](#) describes the corresponding function that each component implements.

### 3.2.2.3 OMX\_CommandStateSet

The IL client calls this command to request that the component transition into the state given in nParam. The component shall make the transition between the old state and the new state successfully only if it is a legal transition and all prerequisites for this transition are met. For more information on component states, see Section 3.1.1.2—OMX\_STATETYPE.

If the component successfully transitions to the new state, [it exposes the new internal state \(e.g. if queried via OMX\\_GetState\(\)\) and then it notifies the IL client of the new state via the OMX\\_EventCmdComplete event, indicating OMX\\_CommandStateSet for nData1 and the new state for nData2, and OMX\\_ErrorNone for error code](#). If a state transition fails, the component shall notify the IL client of the error that prevented it via [OMX\\_EventCmdCompleteError event indicating OMX\\_CommandStateSet for nData1, the attempted transition state for nData2 and an appropriate error code for pEventData](#). Relevant errors include but are not limited to the following:

- OMX\_ErrorSameState: The component is already in the state requested.
- OMX\_ErrorIncorrectStateTransition: The transition requested is not legal.
- OMX\_ErrorInsufficientResources: The transition required the allocation of resources and the component failed to acquire the resources.

### 3.2.2.4 OMX\_CommandFlush

[This](#) [The](#) IL client calls this command to flush one or more component ports. nParam specifies the index of the port to flush. If the value of nParam is OMX\_ALL, the component shall flush all ports. [In tunneled cases, the IL client shall always flush both ends of a tunnel.](#)

When the IL client flushes a non-tunneled port, that port shall return all [the buffers it is holding to the IL client using EmptyBufferDone and FillBufferDone \(appropriate for an input port or an output port, respectively\) to return the buffers.](#)

When [the IL client flushes a non-supplier tunneled port, that port flushed input shall return all the buffers it is holding to its tunneled port using port uses EmptyThisBuffer or FillThisBuffer \(appropriate for an input port or an output port, respectively\) to return the buffers to the output port. Output ports shall zero the nFilledLen field of the buffer header when returning buffers as part of a flush operation.](#)

[When the IL client flushes a supplier port, that port shall hold its buffers.](#)

A non-supplier port that is sharing its buffers with another port of the component may need its sharing port to be flushed as well in order to be able to complete the flush command. In that situation, the component shall report an OMX\_EventPortNeedsFlush event with the index of the sharing port to indicate to the IL client that it needs to flush another port in order for the current flush command to complete.

When a port is flushed, the component shall reset any internal state associated with the port so as to be ready to process from another location within the stream after the flush. This includes discarding any unprocessed data in the queued buffers for a supplier port.

The codec configuration shall not be reset, but the component shall be able to handle the first buffer after the flush containing a new CODEC\_CONFIG indicated by the OMX\_BUFFERFLAG\_CODECCONFIG buffer flag.

For each port that the component successfully flushes, the component shall send an OMX\_EventCmdComplete event, indicating OMX\_CommandFlush for nData1, and the individual port index for nData2, even if the flush resulted from using a value of OMX\_ALL for nParam. and OMX\_ErrorNone for pEventData in case of success or an appropriate error code for pEventData in case of failure. If a flush fails, the component shall notify the IL client of the error via an OMX\_EventError event.

### 3.2.2.5 OMX\_CommandPortDisable

The OMX\_CommandPortDisable command disables a port. nParam specifies the index of the port to disable. If the value of nParam is OMX\_ALL, the component shall disable all ports. In tunneled cases, the IL Client IL client shall always disable both ends of the tunnel.

A disabled port has no buffers and is not connected to either the IL client or another port via a tunnel. A disabled port does not allocate buffers or buffer headers on a transition from OMX\_StateLoaded or OMX\_StateWaitForResources to OMX\_StateIdle. An IL client can change the parameters via OMX\_SetParameter of a disabled port or set up a tunnel on it regardless of the component state. Thus the OMX\_CommandPortDisable command, in co-operation with OMX\_TearDownTunnel, OMX\_SetUpTunnel and OMX\_CommandPortEnable, is useful for the dynamic reconfiguration or re-tunneling of a port.

The port shall immediately clear bEnabled in its port definition structure when it receives OMX\_CommandPortDisable. If the port that the IL client is disabling is a non-supplier port, the component IL client shall return any buffers it is holding to its the supplier port via OMX\_EmptyThisBuffer/OMX\_FillThisBuffer if tunneling or EmptyBufferDone/FillBufferDone if not tunneling. Then, the component IL client shall wait for the supplier port to free the buffers via OMX\_FreeBuffer before completing the disable command. If the port that the IL client is disabling is a supplier port with buffers allocated, the component IL client shall wait for the non-supplier port to return all buffers via OMX\_EmptyThisBuffer or OMX\_FillThisBuffer. Then,

the **componentIL-client** shall free the buffers via OMX\_FreeBuffer before completing the disable command.

For each port that the component **successfully** disables, the component shall send an OMX\_EventCmdComplete event indicating OMX\_CommandPortDisable for nData1, **and** the individual port index for nData2, even if using a value of OMX\_ALL for nParam caused the port to be disabled, **and OMX\_ErrorNone for pEventData in case of success or an appropriate error code for pEventData in case of failure.** If the disable operation fails, the component shall notify the IL client of the error via the OMX\_EventError event.

**A non-supplier port that is sharing its buffers with another port of the component may require its sharing port to be disabled as well in order to be able to complete the OMX\_CommandPortDisable command. In this situation, the component shall report an OMX\_EventPortNeedsDisable event with the index of the sharing port to indicate to the IL client that it needs to disable another port in order for the current OMX\_CommandPortDisable command to complete.**

### 3.2.2.6 OMX\_CommandPortEnable

The OMX\_CommandPortEnable command enables a port. nParam specifies the index of the port to be enabled. If the value of nParam is OMX\_ALL, the component shall enable all ports. **In tunneled cases, the IL client shall always enable both ends of a tunnel.**

An enabled port shall abide by all the requirements of the component's state. Thus, the port shall:

- Have no buffers allocated if the component is in the OMX\_StateLoaded state or the OMX\_StateWaitForResources state and all buffers are allocated otherwise.
- Allocate buffers on a transition from either the OMX\_StateLoaded state or the OMX\_StateWaitForResources state to the OMX\_StateIdle.
- Transfer a buffer to facilitate data flow in the OMX\_StateExecuting state.
- Disallow modification of its parameters via OMX\_SetParameter in all states but OMX\_StateLoaded.

The OMX\_CommandPortEnable command, in co-operation with OMX\_CommandPortDisable, is useful for the dynamic reconfiguration or re-tunneling of a port.

The port shall immediately set bEnabled in its port definition structure when the port receives OMX\_CommandPortEnable. If the IL client enables a port while the component is in any state other than OMX\_StateLoaded or OMX\_StateWaitForResources, then that port shall allocate its buffers via the same call sequence used on a transition from OMX\_StateLoaded to OMX\_StateIdle. If

the IL client enables while the component is in the OMX\_StateExecuting state, then that port shall begin transferring buffers.

For each port that the component successfully enables, the component shall send an OMX\_EventCmdComplete event, indicating OMX\_CommandPortEnable for nData1, and the individual port index for nData2, even if using the value of OMX\_ALL for nParam caused the enable operation, and OMX\_ErrorNone for pEventData in case of success or an appropriate error code for pEventData in case of failure. If a port enablement operation fails, the component shall notify the IL client of the error via OMX\_EventError event.

### 3.2.2.7 OMX\_CommandMarkBuffer

The OMX\_CommandMarkBuffer command instructs the given port to mark a buffer. nParam holds the index of the port that will perform the mark. The pCmdData parameter of OMX\_SendCommand points to an OMX\_MARKTYPE structure. The pMarkTargetComponent field of this structure holds a pointer to the component that will send an event after processing the marked buffer. The pMarkData field of this structure holds a pointer to application-specific data associated with the mark to uniquely identify the mark to the application upon a mark event (denoted the *mark data*).

When instructed to mark a buffer, the component will mark the next buffer that it receives as input after it receives the mark command. The exception is a source component, which will mark the next buffer it adds to its output buffer queue. For components other than source components, the port index value in nParam holds the index of the input port that will mark its next buffer. For source components, the port index value in nParam holds the index of the output port that will mark its next buffer.

In the following cases, multiple marks may compete for a single buffer:

- A component receives two or more mark commands with no intervening buffer(s).
- Two or more input buffers, each with a mark, contribute to an output buffer (e.g., in a mixer).
- A component receives a mark command and the next buffer is already marked.

If multiple marks compete for application to the same buffer, the component uses the first mark received to mark the buffer and applies the remaining marks to subsequent buffers in the order that the component received them. If there are no subsequent buffers, the component may send the remaining marks on one or more empty buffers.

For each case where the component successfully marks a buffer, the component shall send an OMX\_EventCmdComplete event indicating OMX\_CommandMarkBuffer for nData1, and the individual port index for nData2, and OMX\_ErrorNone for pEventData. If a mark operation fails, the component shall notify the IL client of the error via OMX\_EventCmdComplete event with an appropriate error code for pEventData.

A buffer header includes pMarkTargetComponent and the pMarkData fields, whose meaning is identical to those in OMX\_MARKTYPE. A component marks a buffer by copying pMarkTargetComponent and the pMarkData fields from the mark command to the buffer headers. Both fields are NULL by default (i.e., before the buffer being marked). A component propagates the mark fields from an input buffer to an output buffer according to the buffer metadata rules established for buffer flags and timestamps. The target component does not propagate the mark but instead clears both fields to NULL.

When a component receives a buffer, it shall compare its own pointer to the pMarkTargetComponent. If the pointers match, the component shall send a mark event, including pMarkData as a parameter, immediately after the buffer exits the component or has been completely processed in the case where it does not exit the component.

[A component shall accept an OMX\\_CommandMarkBuffer request when in the OMX\\_StateExecuting or OMX\\_StatePause states.](#)

If the port associated with the OMX\_CommandMarkBuffer request is disabled or the component is in the OMX\_StatePause state, the component shall queue the requests and commence processing these queued requests when in the OMX\_StateExecuting state and the associated port is enabled.

If a component has queued OMX\_CommandMarkBuffer requests upon transitioning to OMX\_StateIdle, it shall automatically flush these queued requests - complete the command processing. The component shall emit a failed OMX\_EventCmdComplete callback (with an appropriate error code for pEventData) as the completion status for each flushed queued request being flushed.

OMX\_MARKTYPE is defined as follows.

```
typedef struct OMX_MARKTYPE {
    OMX_HANDLETYPE hMarkTargetComponent;
    OMX_PTR pMarkData;
} OMX_MARKTYPE;
```

The parameters are described as follows.

Parameter	Description
<i>hMarkTargetComponent</i>	Identifies the component handle that shall generate a mark event upon process the mark.
<i>nMarkData</i>	Application specific data associated with mark sent on a mark event to disambiguate a mark from others.

### 3.2.2.7.1 Prerequisites for This Method

This method has no prerequisites.

### 3.2.2.7.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* instructs a component port to mark a buffer*/  
  
OMX_MARKTYPE mark;  
  
mark.hMarkTargetComponent = hComp;  
mark.pMarkData = appData;  
  
OMX_SendCommand(hComp, OMX_CommandMarkBuffer, portIndex, &mark);
```

### 3.2.2.8 OMX\_GetParameter

The OMX\_GetParameter macro will get a parameter setting from a component. The nParamIndex parameter indicates which structure is requested from the component. The caller shall provide memory for the structure and populate the nSize and nVersion fields before invoking this macro. If the parameter settings are for a port, the caller shall also provide a valid port number in the nPortIndex field before invoking this macro. All components shall support a set of defaults for each parameter so that the caller can obtain the structure populated with valid values.

This call is a blocking call. The component should return from this call within 20 milliseconds.

The OMX\_GetParameter macro is defined as follows.

```
#define OMX_GetParameter (   
    hComponent,  
    nParamIndex,  
    pComponentParameterStructure)  
    ((OMX_COMPONENTTYPE*)hComponent)->GetParameter( \\\r  
        hComponent,  
        nParamIndex,  
        pComponentParameterStructure) \\\r  
        \\\r
```

The parameters are described as follows.

Parameter	Description
<i>hComponent</i> [in]	The handle of the component that executes the call
<i>nParamIndex</i> [in]	The index of the structure to be filled. This value is from the OMX_INDEXTYPE enumeration.
<i>pComponentParameterStructure</i> [in,out]	A pointer to the IL client-allocated structure that the component fills

Section 3.3—[OpenMAX IL Component Methods and Structures](#)  
[OpenMAX IL Component Methods and Structures](#)  
[OpenMAX IL Component Methods and Structures](#)  
describes the corresponding function that each component implements.

### **3.2.2.8.1 Prerequisites for This Method**

This method has no prerequisites.

~~The macro can be invoked when the component is in any state except the OMX\_StateInvalid state.~~

### **3.2.2.8.2 Sample Code Showing Calling Sequence**

The following sample code shows the calling sequence.

```
/* disable every audio port of a component*/
OMX_GetParameter(hComp, OMX_IndexParamAudioInit, &oParam);
for (i=0;i<oParam.nPorts;i++) {
    OMX_SendCommand(
        hComp,
        OMX_CommandPortDisable,
        oParam.nStartPortNumber + i,
        0);
}
```

### **3.2.2.8.3 Error Conditions**

The following error conditions can occur:

- OMX\_ErrorBadParameter if one or more fields of the parameter structure are incorrect.
- OMX\_ErrorUnsupportedIndex when the specified parameter index is unsupported.
- OMX\_ErrorVersionMismatch when the nVersion field of the parameter structure does not match the expected version for the component.
- OMX\_ErrorNotReady if an OMX\_GetParameter operation has not completed processing. The caller should retry the OMX\_GetParameter call.
- OMX\_ErrorNoMore when the OMX\_GetParameter function is called with a structure that includes the nPortIndex field and the value of nPortIndex exceeds the number of ports (of the appropriate domain) for the component.

### **3.2.2.9 OMX\_SetParameter**

The OMX\_SetParameter macro will send a parameter structure to a component. The nParamIndex parameter indicates which structure is passed to the component.

The caller shall provide the memory for the correct structure and shall fill in the structure nSize and nVersion fields in addition to all other fields before invoking this macro. The caller is free to dispose of this structure after the call, as the component is required to copy any data it shall retain.

Some parameter structures contain read-only fields. The OMX\_SetParameter method will preserve read-only fields, and shall not generate an error when the caller attempts to change the value of a read-only field.

This call is a blocking call. The component should return from this call within 20 milliseconds.

The OMX\_SetParameter macro is defined as follows.

```
#define OMX_SetParameter ( \
    hComponent, \
    nParamIndex, \
    pComponentParameterStructure) \
    ((OMX_COMPONENTTYPE*)hComponent)->SetParameter( \
        hComponent, \
        nParamIndex, \
        pComponentParameterStructure)
```

The parameters are as follows.

Parameter	Description
<i>hComponent</i> [in]	The handle of the component that executes the call.
<i>nParamIndex</i> [in]	The index of the structure that is to be sent. This value is from the OMX_INDEXTYPE enumeration.
<i>pComponentParameterStructure</i> [in]	A pointer to the IL client-allocated structure that the component uses for initialization.

Section 3.3.6 below describes the corresponding function that each component implements.

### 3.2.2.9.1 Prerequisites for This Method

The OMX\_SetParameter macro can be invoked only when the component is in the OMX\_StateLoaded state or on a port that is disabled.

The only exception to this prerequisite is when the component is transitioning from OMX\_StateLoaded to OMX\_StateIdle, or the port is being enabled. In these two cases, the component shall accept calls to OMX\_SetParameter with the OMX\_IndexParamPortDefinition index. This is needed to modify the value of any of the writable fields in the OMX\_PARAM\_PORTDEFINITIONTYPE structure on a non-supplier port, before any buffers have been allocated on that port. After the first buffer is allocated, OMX\_SetParameter shall no longer be accepted by the port.

### 3.2.2.9.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* force a port to be the supplier */ \
OMX_GetParameter(hComp, OMX_IndexParamPortDefinition, &oPortDef); \
if (oPortDef.eDir == OMX_DirInput){
```

```

        oSupplier.eBufferSupplier = OMX_BufferSupplyInput;
    } else {
        oSupplier.eBufferSupplier = OMX_BufferSupplyOutput;
    }
oSupplier.nPortIndex = nPortIndex;
OMX_SetParameter(hComp, OMX_IndexParamCompBufferSupplier, &oSupplier);

```

### 3.2.2.9.3 Error Conditions

The following error conditions can occur:

- OMX\_ErrorIncorrectStateOperation when the OMX\_SetParameter function is called and the component is not in the OMX\_StateLoaded state, or the port is not disabled.
- OMX\_ErrorBadParameter if one or more fields of the parameter structure are incorrect.
- OMX\_ErrorUnsupportedIndex when the specified parameter index is unsupported.
- OMX\_ErrorVersionMismatch when the nVersion field of the parameter structure does not match the expected version for the component.
- OMX\_ErrorUnsupportedSetting when a field in the parameter structure is unsupported by the component during an OMX\_SetParameter call.
- OMX\_ErrorNotReady if an OMX\_SetParameter operation has not completed processing. The caller should retry the OMX\_SetParameter call.
- ~~OMX\_ErrorNoMore when the OMX\_SetParameter function is called with a structure that includes the nPortIndex field and the value of nPortIndex exceeds the number of ports (of the appropriate domain) for the component.~~

### 3.2.2.10 OMX\_GetConfig

The OMX\_GetConfig macro will get a configuration structure from a component. This macro can be invoked at any time after the component has been loaded. The nConfigIndex parameter indicates which structure is being requested from the component. The caller shall provide the memory for the structure and populate the nSize and nVersion fields before invoking this macro. If the configuration settings are for a port, the caller shall also provide a valid port number in the nPortIndex field before invoking this macro. All components shall support a set of defaults for each configuration so that the caller can obtain the structure populated with valid values.

This call is a blocking call. The component should return from this call within five milliseconds.

The OMX\_GetConfig macro is defined as follows.

```
#define OMX_GetConfig (
    hComponent,
```

```

    nConfigIndex,
    pComponentConfigStructure)
((OMX_COMPONENTTYPE*)hComponent)->GetConfig(
    hComponent,
    nConfigIndex,
    pComponentConfigStructure)

```

The parameters are as follows.

Parameters	Description
<i>hComponent</i> [in]	The handle of the component that executes the call.
<i>nConfigIndex</i> [in]	The index of the structure to be filled. This value is from the OMX_INDEXTYPE enumeration.
<i>pComponentConfigStructure</i> [in,out]	A pointer to the IL client-allocated structure that the component fills.

Section 3.3.7 below describes the corresponding function that each component implements.

### 3.2.2.10.1 Prerequisites for This Method

This method has no prerequisites.

~~The macro can be invoked when the component is in any state except the OMX\_StateInvalid state.~~

### 3.2.2.10.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```

/* Wait until a certain playback position */
do {
    OMX_GetConfig(hClockComp, OMX_IndexConfigTimeCurrentMediaTime,
                  oMediaTime);
} while (oMediaStamp.nTimeStamp < nTargetTimeStamp);

```

### 3.2.2.10.3 Error Conditions

The following error conditions can occur:

- OMX\_ErrorBadParameter if one or more fields of the config structure are incorrect.
- OMX\_ErrorUnsupportedIndex when the specified config index is unsupported.
- OMX\_ErrorVersionMismatch when the nVersion field of the config structure does not match the expected version for the component.

- OMX\_ErrorNotReady if an OMX\_GetConfig operation has not completed processing. The caller should retry the OMX\_GetConfig call.
- OMX\_ErrorNoMore when the OMX\_GetConfig function is called with a structure that includes the nPortIndex field and the value of nPortIndex exceeds the number of ports (of the appropriate domain) for the component.

### 3.2.2.11 OMX\_SetConfig

The OMX\_SetConfig macro will set a component configuration value. This macro can be invoked anytime after the component has been loaded.

The caller shall provide the memory for the correct structure and fill in the structure nSize and nVersion fields in addition to all other fields before invoking this macro. The caller can dispose of this structure after the call, as the component is required to copy any data it shall retain.

Some configuration structures contain read-only fields. The OMX\_SetConfig method will preserve read-only fields in configuration structures that contain them, and shall not generate an error when the caller attempts to change the value of a read-only field.

This call is a blocking call. The component should return from this call within five milliseconds.

The OMX\_SetConfig macro is defined as follows.

```
#define OMX_SetConfig ( \
    hComponent, \
    nConfigIndex, \
    pComponentConfigStructure \
) ((OMX_COMPONENTTYPE*)hComponent)->SetConfig( \
    hComponent, \
    nConfigIndex, \
    pComponentConfigStructure)
```

The parameters are as follows.

Parameter	Description
<i>hComponent</i>	The handle of the component that executes the call. [in]
<i>nConfigIndex</i>	The index of the structure that is to be sent. This value is from the OMX_INDEXTYPE enumeration. [in]
<i>pComponentConfigStructure</i>	A pointer to the IL client-allocated structure that the component uses for initialization. [in]

Section 3.3.8 below describes of the corresponding function that each component implements.

#### 3.2.2.11.1 Prerequisites for This Method

[This method has no prerequisites.](#)

~~The macro can be invoked when the component is in any state except the OMX\_StateInvalid state.~~

### 3.2.2.11.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* Change the time scale of the clock component*/
oScale.xScale = 0x00020000; /*2x*/
OMX_SetConfig(hClockComp, OMX_IndexConfigTimeScale, (OMX_PTR)&oScale);
```

### 3.2.2.11.3 Error Conditions

The following error conditions can occur:

- OMX\_ErrorBadParameter if one or more fields of the config structure are incorrect.
- OMX\_ErrorUnsupportedIndex when the specified config index is unsupported.
- OMX\_ErrorVersionMismatch when the nVersion field of the config structure does not match the expected version for the component.
- OMX\_ErrorUnsupportedSetting when a field in the config structure is unsupported by the component during an OMX\_SetConfig call.
- OMX\_ErrorNotReady if an OMX\_SetConfig operation has not completed processing. The caller should retry the OMX\_SetConfig call.
- ~~OMX\_ErrorNoMore when the OMX\_SetConfig function is called with a structure that includes the nPortIndex field and the value of nPortIndex exceeds the number of ports (of the appropriate domain) for the component.~~

## 3.2.2.12 OMX\_GetExtensionIndex

The OMX\_GetExtensionIndex macro will invoke a component to ~~convert translate from a standardized OpenMAX IL or vendor-specific extension string for a configuration or a parameter into~~ an OpenMAX IL ~~configuration or parameter structure~~ index. The vendor is not required to support this command for the indexes already found in the OMX\_INDEXTYPE enumeration, which reduces the memory footprint. The component may support any standardized OpenMAX IL or vendor-specific extension indexes that are not found in the master OMX\_INDEXTYPE enumeration.

This call is a blocking call. The component should return from this call within five milliseconds.

The OMX\_GetExtensionIndex macro is defined as follows.

```
#define OMX_GetExtensionIndex (
    hComponent,
    cParameterName,
```

```

    pIndexType      )
((OMX_COMPONENTTYPE*)hComponent)->GetExtensionIndex(      \
    hComponent,          \
    cParameterName,      \
    pIndexType)          \

```

The parameters are as follows.

Parameter	Description
<i>hComponent</i> [in]	The handle of the component that executes the call.
<i>cParameterName</i> [in]	An OMX_STRING value that shall be less than 128 characters long including the trailing null byte. The component will translate this string into a configuration index.
<i>pIndexType</i> [out]	A pointer to the OMX_INDEXTYPE structure that is to receive the index value.

Section 3.3.9 below describes the corresponding function that each component implements.

### 3.2.2.12.1 Prerequisites for This Method

This method has no prerequisites.

~~The macro can be invoked when the component is in any state except the OMX\_StateInvalid state.~~

### 3.2.2.12.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```

/* Set the vendor-specific filename parameter on a reader */
OMX_GetExtensionIndex(
    hFileReaderComp,
    "OMX.CompanyXYZ.index.param.filename",
    &eIndexParamFilename);
OMX_SetParameter(hComp, eIndexParamFilename, &oFileName);

```

### 3.2.2.13 OMX\_GetState

The OMX\_GetState macro will invoke the component to get the current state of the component and place the state value into the location pointed to by *pState*. The component should return from this call within five milliseconds.

The OMX\_GetState macro is defined as follows.

```

#define OMX_GetState (
    hComponent,
    pState      )
((OMX_COMPONENTTYPE*)hComponent)->GetState(      \
    hComponent,          \
    pState)          \

```

The parameters are as follows.

Parameter	Definition
<i>hComponent</i>	The handle of the component that executes the call. [in]
<i>pState</i>	A pointer to the location that receives the state. The value returned is one of the [out] OMX_STATETYPE members.

Section 3.3.10 below describes the corresponding function that each component implements.

### **3.2.2.13.1 Prerequisites for This Method**

This method has no prerequisites.

### **3.2.2.13.2 Sample Code Showing Calling Sequence**

The following sample code shows the calling sequence.

```
OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StateIdle, 0);
do {
    OMX_GetState(hComp, &eState);
} while (OMX_StateIdle != eState);
```

### **3.2.2.14 OMX\_UseBuffer**

The OMX\_UseBuffer macro requests the component to use a buffer *already*-allocated by the IL client or a buffer *already* supplied by a tunneled component. The OMX\_UseBuffer implementation shall allocate the buffer header, populate it with the given input parameters, and pass it back via the *ppBufferHdr* output parameter.

When populating fields within the buffer header structure, components are required to correctly *initialiseinitialize* both *pInputPortIndex* and *pOutputPortIndex*. They are also required to *initialiseinitialize* the *pAppPrivate* field with the *pAppPrivate* function parameter. The *pAppPrivate* parameter should also be used to *initialiseinitialize* the *pInputPortPrivate* or *pOutputPortPrivate* field, when called on an output port or input port respectively.

When OMX\_UseBuffer is used for pre-announcing a buffer pointer, the pBuffer and nAllocLen fields in the allocated buffer header shall be initialized with the pBuffer and nSizeBytes function parameters, respectively. When the pBuffer function parameter is NULL, the component shall initialize pBuffer and nAllocLen with NULL and zero, respectively.

The OMX\_UseBuffer macro *canshall* be *called on a componentexecuted* under the following conditions:

- While the component is in the OMX\_StateLoaded state and has already received a command sent a request for the state to transition to the OMX\_StateIdle state
- While the component is in the OMX\_StateWaitForResources state, the resources needed are available, and the component is ready to go to the OMX\_StateIdle state
- On a disabled port when the component is in one of the OMX\_StateExecuting, the OMX\_StatePause, or the OMX\_StateIdle states and has already received a command to enable the port.
- In tunneled case for a non-supplier port, the component shall call OMX\_SetConfig(OMX\_PORTSTATUS\_ACCEPTUSEBUFFER) on its tunneled component to indicate when it can start calling UseBuffer.

This is a blocking call. The component should return from this call within 20 milliseconds.

The OMX\_UseBuffer macro is defined as follows.

```
#define OMX_UseBuffer( \
    hComponent, \
    ppBufferHdr, \
    nPortIndex, \
    pAppPrivate, \
    nSizeBytes, \
    pBuffer) \
    ((OMX_COMPONENTTYPE*)hComponent->UseBuffer( \
        hComponent, \
        ppBufferHdr, \
        nPortIndex, \
        pAppPrivate, \
        nSizeBytes, \
        pBuffer))
```

The parameters are as follows.

Parameter	Description
<i>hComponent</i> [in]	The handle of that component that executes the call.
<i>ppBufferHdr</i> [out]	A pointer to a pointer of an OMX_BUFFERHEADERTYPE structure that receives the pointer to the buffer header.
<i>nPortIndex</i> [in]	The index of the port that will use the specified buffer. This index is relative to the component that owns the port.
<i>pAppPrivate</i> [in]	A pointer that refers to an implementation-specific memory area that is under responsibility of the supplier of the buffer.
<i>nSizeBytes</i> [in]	The buffer size in bytes.

Parameter	Description
<i>pBuffer</i> [in]	A pointer to the memory buffer area to be used. <u>When non-NUL</u> , <u>the call will establish a one-to-one relationship between the</u> <u>memory buffer and the buffer header. When NULL, the actual</u> <u>memory buffer is transferred only via OMX_EmptyThisBuffer or</u> <u>OMX_FillThisBuffer calls.</u>

Section 3.3.12 below describes the corresponding function that each component implements.

#### 3.2.2.14.1 Prerequisites for This Method

The component shall be in the OMX\_StateLoaded or the OMX\_StateWaitForResources state, or the port to which the call applies shall be disabled.

#### 3.2.2.14.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* supplier port allocates buffers and pass them to non-supplier */
for (i=0;i<pPort->nBufferCount;i++)
{
    pPort->pBuffer[i] = malloc(pPort->nBufferSize);
    OMX_UseBuffer(pPort->hTunnelComponent,
                  &pPort->pBufferHdr[i],
                  pPort->nTunnelPort,
                  pPort,
                  pPort->nBufferSize,
                  pPort->pBuffer[i]);
}
```

#### 3.2.2.15 OMX\_AllocateBuffer

The OMX\_AllocateBuffer macro will request that the component allocate a new buffer and buffer header. The component will allocate the buffer and the buffer header and return a pointer to the buffer header.

When populating fields within the buffer header structure, components are required to correctly initialize both pInputPortIndex and pOutputPortIndex. They are also required to initialize the pAppPrivate field with the pAppPrivate function parameter. The pAppPrivate parameter should also be used to initialize the pInputPortPrivate or pOutputPortPrivate field, when called on an output port or input port respectively.

The OMX\_AllocateBuffer macro can be called on a component This call is a blocking call that shall be performed under the following conditions:

- While the component is in the OMX\_StateLoaded state and has already received a command sent a request for the state to transition to OMX\_StateIdle.

- While the component is in the OMX\_StateWaitForResources state, the resources needed are available, and the component is ready to go to the OMX\_StateIdle state.
- On a disabled port when the component is in one of the OMX\_StateExecuting, the OMX\_StatePause, or the OMX\_StateIdle states and has already received a command to enable the port.

The OMX\_AllocateBuffer macro allocates buffers on a specific port for communication with the IL client only. This macro cannot be used to allocate buffers for tunneled ports. Buffers allocated before a port was configured for tunneling will result in the component failing OMX\_SetupTunnel calls to the port.

The component should return from this call within five milliseconds.

The OMX\_AllocateBuffer macro is defined as follows.

```
#define OMX_AllocateBuffer( \
    hComponent, \
    ppBuffer, \
    nPortIndex, \
    pAppPrivate, \
    nSizeBytes) \
    ((OMX_COMPONENTTYPE*)hComponent)->AllocateBuffer( \
        hComponent, \
        ppBuffer, \
        nPortIndex, \
        pAppPrivate, \
        nSizeBytes)

#define OMX_AllocateBuffer( \
    hComponent, \
    pBuffer, \
    nPortIndex, \
    pAppPrivate, \
    nSizeBytes) \
    ((OMX_COMPONENTTYPE*)hComponent)->AllocateBuffer( \
        hComponent, \
        pBuffer, \
        nPortIndex, \
        pAppPrivate, \
        nSizeBytes)
    \
    ((OMX_COMPONENTTYPE*)hComponent)->AllocateBuffer( \
        hComponent, \
        ppBuffer, \
        nPortIndex, \
        pAppPrivate, \
        nSizeBytes)
    \
    pAppPrivate, \
    nSizeBytes, \
    pBuffer)
```

The parameter are as follows.

Parameter	Description
<i>hComponent</i> [in]	The handle of the component that executes the call.
<i>ppBuffer</i> [out]	A pointer to a pointer of an OMX_BUFFERHEADERTYPE structure that receives the pointer to the buffer header.

<b>Parameter</b>	<b>Description</b>
<i>nPortIndex</i> [in]	Selects the port on the component that the buffer will be used with. The port can be found by using the <i>nPortIndex</i> value as an index into the port definition array of the component.
<i>pAppPrivate</i> [in]	Initializes the <i>pAppPrivate</i> member of the buffer header structure.
<i>nSizeBytes</i> [in]	The size of the buffer to allocate.

Section 3.3.13 below describes the corresponding function that each component implements.

### 3.2.2.15.1 Prerequisites for This Method

The component shall be in the OMX\_StateLoaded or the OMX\_StateWaitForResources state, or the port to which the call applies shall be disabled.

### 3.2.2.15.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* IL client asks component to allocate buffers */
for (i=0;i<pClient->nBufferCount;i++)
{
    OMX_AllocateBuffer(hComp,
                        &pClient->pBufferHdr[i],
                        pClient->nPortIndex,
                        pClient,
                        pClient->nBufferSize);
}
```

## 3.2.2.16 OMX\_FreeBuffer

The OMX\_FreeBuffer macro will release a buffer and buffer header from the component. The component shall free only the buffer header if it allocated only the buffer header. The component shall free both the buffer and the buffer header if it allocated both the buffer and the buffer header. Thus, the component shall track which buffers it allocated so it can perform the corresponding de-allocation.

The call should be performed under the following conditions:

- While the component is in the OMX\_StateIdle state and the IL client has already sent a request for the state transition to OMX\_StateLoaded (e.g., during the stopping of the component)
- On a disabled port when the component is in the OMX\_StateExecuting, the OMX\_StatePause, or the OMX\_StateIdle state.

The call can be made at any time [provided the caller owns the buffer](#), but may result in the port sending an `OMX_ErrorPortUnpopulated` event error if the call is not performed as described.

The call is made from **buffer suppliers** ~~ports when tunneling~~ to release buffer headers from ~~non-supplier ports~~ the port that the supplier port is tunneling with.

This call is a blocking call. The component should return from the call within 20 milliseconds.

The OMX\_FreeBuffer macro is defined as follows:

```
#define OMX_FreeBuffer ( hComponent, nPortIndex, pBuffer ) ((OMX_COMPONENTTYPE*)hComponent)->FreeBuffer( hComponent, nPortIndex, pBuffer )  
pBuffer )
```

The parameters are as follows.

Parameter	Description
<i>hComponent</i> [in]	The handle of the component that executes the call
<i>nPortIndex</i> [in]	The index of the port that is using the specified buffer
<i>pBuffer</i> [in]	A pointer to an OMX_BUFFERHEADERTYPE structure used to provide or receive the pointer to the buffer header.

Section 3.3.14 describes the corresponding function that each component implements.

### **3.2.2.16.1 Prerequisites for This Method**

The component should be in the OMX\_StateIdle state or the port should be disabled.

### **3.2.2.16.2 Sample Code Showing Calling Sequence**

The following sample code shows the calling sequence.

```
/* supplier port frees buffers */
for (i=0;i<pPort->nBufferCount;i++)
{
    free(pPort->pBuffer[i]);
    pPort->pBuffer[i] = 0;
    OMX_FreeBuffer(pPort->hTunnelComponent,
                    pPort->nTunnelPort,
                    pPort->pBufferHdr[i]);
    pPort->pBufferHdr[j] = 0;
}
```

### 3.2.2.17 OMX\_EmptyThisBuffer

The OMX\_EmptyThisBuffer macro will send a filled buffer to an input port of a component. When the buffer contains data, the value of the nFilledLen field of the buffer header will not be zero. If the buffer contains no data, the value of nFilledLen is 0x0. The OMX\_EmptyThisBuffer macro shall be executed is invoked to pass buffers containing data when the component is in one of or making a transition to the OMX\_StateExecuting or in the OMX\_StatePause states. In tunneled case, a component shall call OMX\_SetConfig(OMX\_PORTSTATUS\_ACCEPTBUFFEREXCHANGE) on its tunneled component to indicate when it can start exchanging buffers.

When a port is non-tunneled, buffers sent to using OMX\_EmptyThisBuffer are returned to the IL client with the EmptyBufferDone callback once they have been emptied.

When a port is tunneled, buffers sent to using OMX\_EmptyThisBuffer are can be returned using OMX\_FillThisBuffer, sent to the tunneled port once they are emptied so long as the component is in the OMX\_StateExecuting state. Buffers are returned to the input port that supplied them using OMX\_EmptyThisBuffer whenever the tunneled port is flushed or disabled. Buffers are also returned to the input port that supplied them when the component calling OMX\_FillThisBuffer is transitioning from the OMX\_StateExecuting state or the OMX\_StatePause state to the OMX\_StateIdle state.

Formatted: Font: (Default) Courier New

This call is a non-blocking call since the component will queue the buffer and return immediately. The buffer will be emptied later at the proper time. If the parameter nInputPortIndex in the buffer header does not specify a valid input port, the component returns OMX\_ErrorBadPortIndex. The component should return from this call within five milliseconds.

The OMX\_EmptyThisBuffer macro is defined as follows.

```
#define OMX_EmptyThisBuffer ( \
    hComponent, \
    pBuffer      ) \
    ((OMX_COMPONENTTYPE*)hComponent)->EmptyThisBuffer( \
        hComponent, \
        pBuffer)
```

The parameters are as follows.

Parameter	Description
<i>hComponent</i> [in]	The handle of the component that executes the call.
<i>pBuffer</i> [in]	A pointer to an OMX_BUFFERHEADERTYPE structure that is used to provide or receive the pointer to the buffer header. The buffer header shall specify the index of the input port that receives the buffer

Section 3.3.15 below describes the corresponding function that each component implements.

### 3.2.2.17.1 Prerequisites for This Method

The component shall be in the appropriate state as shown in [Table 3-17](#).

### 3.2.2.17.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* deliver full buffer */
if (pPort->hTunnelComponent)
OMX_EmptyThisBuffer(pPort->hTunnelComponent, pBuffer);
else
    pCallbacks->FillBufferDone(hComp, pBuffer,
pPort->pCallbackAppData);
```

## 3.2.2.18 OMX\_FillThisBuffer

The OMX\_FillThisBuffer macro will send an empty buffer to an output port of a component. The OMX\_FillThisBuffer macro ~~shall be executed~~ is invoked to pass buffers containing no data when the component is in ~~one of or making a transition to~~ the OMX\_StateExecuting state or ~~is in the~~ OMX\_StatePause states. In tunneled case, a component shall call OMX\_SetConfig(OMX\_PORTSTATUS\_ACCEPTBUFFEREXCHANGE) on its tunneled component to indicate when it can start exchanging buffers.

When a port is non-tunneled, buffers sent ~~to using~~ OMX\_FillThisBuffer return to the IL client with the FillBufferDone callback ~~once they have been filled~~.

When a port is tunneled, buffers sent ~~using~~ OMX\_FillThisBuffer ~~can be returned are sent to the tunneled port once they are filled so long as the component is in the OMX\_StateExecuting state. Buffers are returned to the output port that supplied them using OMX\_FillThisBufferEmptyThisBuffer whenever the tunneled port is flushed or disabled. Buffers are also returned to the output port that supplied them when the component that calls OMX\_FillThisBuffer is transitioning from the OMX\_StateExecuting state or OMX\_StatePause state to the OMX\_StateIdle state.~~

This call is a non-blocking call since the component will queue the buffer and return immediately. The buffer will be filled later at the proper time. If the parameter nOutputPortIndex in the buffer header does not specify a valid output port, the component returns OMX\_ErrorBadPortIndex. The component should return from this call within five milliseconds.

The OMX\_FillThisBuffer macro is defined as follows.

```
#define OMX_FillThisBuffer (
    hComponent,
    pBuffer
)
```

```
( (OMX_COMPONENTTYPE*)hComponent )->FillThisBuffer(           \
    hComponent,                                              \
    pBuffer)
```

The parameters are as follows.

**Parameter**

**Description**

<i>hComponent</i>	The handle of the component that executes the call. [in]
<i>pBuffer</i>	A pointer to an OMX_BUFFERHEADERTYPE structure used to provide or receive the pointer to the buffer header. The buffer header shall specify the index of the <i>output<input/></i> port that receives the buffer. [in]

Section 3.3.16 below describes the corresponding function that each component implements.

### 3.2.2.18.1 Prerequisites for This Method

The component shall be in the appropriate state as shown in [Table 3-17](#).

### 3.2.2.18.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* On a port enable, if tunneling and an input and not supplier */
/* then give buffers to supplier port */
if (pPort->hTunnelComponent &&
    (pPort->oPortDef.eDir == OMX_DirInput) &&
    (pPort->eSupplierSetting == OMX_BufferSupplyInput) )
{
    for (i=0;i<pPort->nBuffers;i++)
        OMX_FillThisBuffer(pPort->hTunnelComponent,
                           pPort->ppBufferHdtrs[i]);
}
```

### 3.2.2.19 OMX\_UseEGLImage

OMX\_UseEGLImage enables an OMX IL component to use as a buffer, the image already allocated via EGL. EGLImages are designed for sharing data between rendering based EGL interfaces, such as OpenGL ES and OpenVG. The format of an EGLImage is opaque to the EGL's client by design, so any memory allocated through this macro are not accessible directly by the IL client.

A method for this interface shall be provided by the component, but may not be implemented, by returning OMX\_ErrorNotImplemented. Components should inspect the EGLImage provided to the method, and determine if the EGLImage is compatible with the port configuration.

The `OMX_UseEGLImage` macro requests that the component use an EGLImage provided by EGL, in place of using the `OMX_UseBuffer` method. The

`OMX_UseEGLImage` implementation shall allocate the buffer header, populate it with the given input parameters, and pass it back via the `ppBufferHdr` output parameter. The `pBuffer` field of the `pBufferHdr` parameter shall be `0x0`, because the format of the EGLImage is opaque to the [IL client](#).

The `OMX_UseEGLImage` macro shall be executed under the following conditions:

- While the component is in the `OMX_StateLoaded` state and has already sent a request for the state transition to `OMX_StateIdle`.
- While the component is in the `OMX_StateWaitForResources` state, the resources needed are available, and the component is ready to go to the `OMX_StateIdle` state.
- On a disabled port when the component is in the `OMX_StateExecuting`, the `OMX_StatePause`, or the `OMX_StateIdle` state.

This is a blocking call. The component should return from this call within 20 milliseconds.

The `OMX_UseEGLImage` macro is defined as follows.

```
#define OMX_UseEGLImage( \
    hComponent, \
    ppBufferHdr, \
    nPortIndex, \
    pAppPrivate, \
    eglImage) \
((OMX_COMPONENTTYPE*)hComponent->UseEGLImage( \
    hComponent, \
    ppBufferHdr, \
    nPortIndex, \
    pAppPrivate, \
    eglImage))
```

The parameters are as follows.

Parameter	Description
<code>hComponent</code>	The handle of that component that executes the call. [in]
<code>ppBufferHdr</code>	A pointer to a pointer of an <code>OMX_BUFFERHEADERTYPE</code> structure [out] that receives the pointer to the buffer header.
<code>nPortIndex</code>	The index of the port that will use the specified buffer. This index [in] is relative to the component that owns the port.
<code>pAppPrivate</code>	A pointer that refers to an implementation-specific memory area [in] that is under responsibility of the supplier of the buffer.
<code>eglImage</code>	The handle of the EGLImage to use as a buffer on the [in] specified port. The component is expected to validate properties of the EGLImage against the configuration of the port to ensure the component can use the EGLImage as a buffer.

Section 3.3.19 below describes the corresponding function that each component implements.

### **3.2.2.19.1 Prerequisites for This Method**

The component shall be in the OMX\_StateLoaded or the OMX\_StateWaitForResources state, or the port to which the call applies shall be disabled.

### 3.2.2.20 OMX SetCallbacks

The `OMX_SetCallbacks` macro will transfer new callbacks information from the IL client to the component. The `OMX_SetCallbacks` macro is invoked to pass a pointer to an `OMX_CALLBACKTYPE` structure containing the callbacks that the component will use for this IL client. Also a pointer to an IL client-defined value is passed. This value shall be returned to the IL client during callbacks so that the IL client can determine the context and/or the source of the callback.

In addition, the component shall update the pApplicationPrivate field of the OMX\_COMPONENTTYPE structure during the call to OMX\_SetCallbacks.

The component shall guarantee that once that this method returns, the pointers for any previously existing callbacks and IL client-defined values will no longer be used in future callbacks.

**This call is a blocking call. The component should return from this call within 20 milliseconds.**

The `OMX_SetCallbacks` macro is defined as follows.

```
#define OMX_SetCallbacks (\  
    hComponent,\  
    pCallbacks,\  
    pAppData)\  
    ((OMX_COMPONENTTYPE*)hComponent)->SetCallbacks(  
        hComponent,  
        pCallbacks,  
        pAppData)
```

The parameters are as follows.

<u>Parameter</u>	<u>Description</u>
<i>hComponent</i> <i>[in]</i>	The handle of the component that executes the call.
<i>pCallbacks</i> <i>[in]</i>	A pointer to an OMX_CALLBACKTYPE structure that is used to provide the callback information to the component.
<i>pAppData</i> <i>[in]</i>	A pointer to a value that the IL client has defined (for example, a pointer to a data structure) that allows the callback in the IL client to determine the context of the call.

### **3.2.2.20.1 Prerequisites for This Method**

The component shall be in the OMX\_StateLoaded state.

### **3.2.2.20.2 Sample Code Showing Calling Sequence**

The following sample code shows the calling sequence.

```
/* On GetHandle (for statically linked components):
   create component, initialize it, and set its callbacks */
pComp = (OMX_COMPONENTTYPE *)malloc(sizeof(OMX_COMPONENTTYPE));
hHandle = (OMX_HANDLETYPE)pComp;
pComp->nVersion = version_1_2_0;
pComp->nSize = sizeof(OMX_COMPONENTTYPE);
OMX_ComponentRegistered[i].pInitialize(hHandle);
pComp->OMX_SetCallbacks(hHandle, pCallBacks, pAppData);
```

## **3.2.3 Functions**

This section describes the functions in the OpenMAX IL API.

### **3.2.3.1 OMX\_Init**

The OMX\_Init method initializes the OpenMAX IL core. ~~Each OpenMAX IL client shall use OMX\_Init as their first call made into OpenMAX IL and this client shall later make a paired OMX\_Deinit call, as the last call into OpenMAX IL. The OpenMAX IL core implementation shall keep track of OMX\_Init and OMX\_Deinit calls. These calls should be executed only one time without an intervening OMX\_Deinit call. If OMX\_Init is called twice, OMX\_ErrorNone is returned but the init request is ignored.~~ The core should return from this call within 20 milliseconds.

The usage of OMX\_Init() is as follows.

```
OMX_API OMX_ERRORTYPE OMX_APIMTRY OMX_Init()
```

#### **3.2.3.1.1 Prerequisites for This Method**

This method has no prerequisites.

#### **3.2.3.1.2 Results/Outputs for This Method**

If the command successfully executes, the return code will be OMX\_ErrorNone. Otherwise, the appropriate OpenMAX IL error will be returned. The OpenMAX IL core functions are ready to be used when this function returns successfully.

#### **3.2.3.1.3 Sample Code Showing Calling Sequence**

The following sample code shows the calling sequence.

```
/* Initialize OpenMAX IL and create some components */
OMX_Init();
```

```

OMX_GetHandle(hMp3Decoder, "OMX.CompanyXYZ.mp3.decoder",
    pAppData, pCallbacks);
OMX_GetHandle(hAudioMixer, "OMX.CompanyXYZ.audio.mixer",
    pAppData, pCallbacks);

```

### 3.2.3.2 OMX\_Deinit

The final OMX\_Deinit callmethod de-initializes the OpenMAX IL core. An OpenMAX IL client shall use OMX\_Deinit asshouldbetheir last call madeintothe OpenMAX IL\_core after all OpenMAX IL-related resources have been released. The client shall only call OMX\_Deinit once per OMX\_Init call. The OpenMAX IL core implementation shall keep track of OMX\_Init and OMX\_Deinit calls. If there are valid component handles when the final call to OMX\_Deinit is made, the OMX\_Deinit shall fail and return OMX\_ErrorIncorrectStateOperation. The core should return from this call within 20 milliseconds. While it may be preferable to have the core command each of the components back to the loaded state and then de-initialize them, doing so may require more than the recommended 20 milliseconds call time. It further requires the OpenMAX IL core to track all component handles, which may add unnecessary complexity for some platforms.

The OMX\_Deinit method usage is as follows.

```
OMX_API OMX_ERRORTYPE OMX_APINTRY OMX_Deinit()
```

#### 3.2.3.2.1 Prerequisites for This Method

The use of OMX\_Deinit requires that all component handles acquired by the IL client in the system have been released, implying that all resources associated with components have been freed.

#### 3.2.3.2.2 Results/Outputs for This Method

The use of OMX\_Deinit returns OMX\_ERRORTYPE. If the command successfully executes, the return code will be OMX\_ErrorNone. Otherwise, the appropriate OpenMAX IL error will return.

#### 3.2.3.2.3 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```

/* Determine if a component of a particular name exists. */
OMX_Init();
eError = OMX_ErrorNone;
for (i=0; OMX_ErrorNone == eError; i++)
{
    eError = OMX_ComponentNameEnum(szCompEnumName, 256, i);
    if ((OMX_ErrorNone == eError) &&
        (!strcmp(szCompEnumName, szComponentName)))
    {
        OMX_Deinit();
        return OMX_TRUE;
    }
}

```

```

    }
OMX_Deinit();
return OMX_FALSE;

```

### 3.2.3.3 OMX\_ComponentNameEnum

The OMX\_ComponentNameEnum method will [allow the IL client to](#) enumerate through all the names of recognized components in the system to detect all the components in the system run-time. There is no strict ordering to the enumeration of component names, although each name shall be enumerated only once. If the OpenMAX IL core supports run-time installation of new components, it is required to detect newly installed components only when the first call to enumerate component names occurs (i.e., when the value of nIndex is 0x0).

The OMX\_ComponentNameEnum method is defined as follows.

```

OMX_API OMX_ERRORTYPE OMX_APientry OMX_ComponentNameEnum(
    OMX_OUT OMX_STRING      cComponentName,
    OMX_IN  OMX_U32          nNameLength,
    OMX_IN  OMX_U32          nIndex
)

```

The parameters are as follows.

Parameter	Description
<i>cComponentName</i> [out]	A pointer to a null-terminated string with the component name. Component names are strings limited to less than 127 bytes in length plus the trailing null for a maximum length of 128 bytes. An example of a valid component name is "OMX.<vendor_name>.AUDIO.DSP.MIXER\0". The name shall start with "OMX." concatenated to a vendor-specified string.
<i>nNameLength</i> [in]	The number of characters in the <i>cComponentName</i> string. Since all component name strings are restricted to less than 128 characters, not including the trailing null, the caller should provide an input string of at least 128 characters.
<i>nIndex</i> [in]	A number containing the enumeration index for the component. Multiple calls to OMX_ComponentNameEnum with increasing values of <i>nIndex</i> will enumerate through the component names in the system until OMX_ErrorNoMore returns. The value of <i>nIndex</i> is 0 to N-1, where N is the number of installed components in the system.

#### 3.2.3.3.1 Prerequisites for This Method

OMX\_ComponentNameEnum can be called after the OMX\_Init function.

#### 3.2.3.3.2 Results/Outputs for This Method

If OMX\_ComponentNameEnum successfully executes, the return code will be OMX\_ErrorNone. When the value of *nIndex* exceeds the number of components in

the system minus 1, OMX\_ErrorNoMore will be returned. Otherwise, the appropriate OpenMAX IL error will be returned.

### 3.2.3.3.3 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* print a list of all components */
eError = OMX_ErrorNone;
for (i=0; OMX_ErrorNoMore != eError; i++)
{
    eError = OMX_ComponentNameEnum(szCompName, 256, i);
    if (OMX_ErrorNone == eError)
        printf("Component %i: %s\n", szCompName);
}
```

### 3.2.3.4 OMX\_GetHandle

The OMX\_GetHandle method will locate the component specified by the component name given, load that component into memory, and validate it. If the component is valid, OMX\_GetHandle will invoke the component's methods to fill the component handle and set up the callbacks. The OMX\_GetHandle method will allocate the actual OMX\_HANDLETYPE structure, ensures it is populated correctly, and then updates the value of \*pHandle with a pointer to the newly created handle. The component should return from this call within 20 milliseconds.

Each time the OMX\_GetHandle function returns successfully, a new component instance is created. The IL client shall configure the newly created component, which is in the OMX\_StateLoaded state, before the component can be used. [After creating a new component all ports shall default to enabled and not connected to any other ports via a tunnel.](#)

Since components are requested by name, a naming convention is defined. OpenMAX IL component names are zero terminated strings with the following format:

“OMX.<vendor\_name>.<vendor\_specified\_convention>”.

For example:

OMX.CompanyABC.MP3Decoder.productXYZ

No standardization among component names is dictated across different vendors.

OMX\_GetHandle is defined as follows.

```
OMX_API OMX_ERRORTYPE OMX_APINTRY OMX_GetHandle(
    OMX_OUT OMX_HANDLETYPE *      pHandle,
    OMX_IN  OMX_STRING           cComponentName,
    OMX_IN  OMX_PTR              pAppData,
    OMX_IN  OMX_CALLBACKTYPE *   pCallBacks
)
```

The parameters are as follows.

Parameter	Description
<i>pHandle</i> [out]	A pointer to OMX_HANDLETYPE to be filled in by this method.
<i>cComponentName</i> [in]	A pointer to a null-terminated string with the component name. Component names are strings limited to less than 128 bytes in length plus the trailing null for a maximum length of 128 bytes. An example of a valid component name is “OMX.<vendor_name>.AUDIO.DSP.MIXER 0”. The name shall start with “OMX.” concatenated to a vendor-specified string.
<i>pAppData</i> [in]	A pointer to an IL client-defined value that will be returned during callbacks so that the IL client can identify the source of the callback.
<i>pCallBacks</i> [in]	A pointer to an OMX_CALLBACKTYPE structure containing the callbacks that the component will use for this IL client.

#### 3.2.3.4.1 Prerequisites for This Method

The OpenMAX IL core shall be initialized.

#### 3.2.3.4.2 Results/Outputs for This Method

If successful, the function returns a valid component handle to the IL client.

#### 3.2.3.4.3 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* determine maximum number of instantiations of a component */
eError = OMX_ErrorNone;
for (i=0; OMX_ErrorNone == eError; i++)
{
    eError = OMX_GetHandle(&hComp[i],
                          szComponentName,
                          pAppData,
                          pCallbacks);
}
printf("Created %i instantiations.\n", i);
```

#### 3.2.3.5 OMX\_FreeHandle

The OMX\_FreeHandle method will free a handle allocated by the OMX\_GetHandle method. The component should return from this call within 20 milliseconds. The IL client should call OMX\_FreeHandle only when the component is in the OMX\_StateLoaded or the OMX\_StateInvalid state and when all the ports are not connected via any tunnels. Calling OMX\_FreeHandle from any other state may result in the component taking longer than the recommended 20 milliseconds execution time, and is provided only as a failure recovery mechanism.

OMX\_FreeHandle is defined as follows.

```
OMX_API OMX_ERRORTYPE OMX_APENTRY OMX_FreeHandle(
```

```
OMX_IN OMX_HANDLETYPE hComponent )
```

The single parameter is as follows.

**Parameter      Description**

*hComponent*    The handle of the component to be freed.  
[in]

#### **3.2.3.5.1 Prerequisites for This Method**

The component should be in the OMX\_StateLoaded ~~or the OMX\_StateInvalid~~ state when this method is called.

#### **3.2.3.5.2 Results/Outputs for This Method**

All resources associated with the components are freed.

#### **3.2.3.5.3 Sample Code Showing Calling Sequence**

The following sample code shows the calling sequence.

```
/* stop executing component and clean up component */
OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StateIdle, 0);
OMX_SendCommand(hComp, OMX_CommandStateSet, OMX_StateLoaded, 0);
do {
    OMX_GetState(hComp, &eState);
} while (OMX_StateLoaded != eState);
OMX_FreeHandle(hComp);
```

#### **3.2.3.6 OMX\_SetupTunnel**

The OMX\_SetupTunnel method sets up tunneled communication between an output port and an input port. This method is an actual method and not a defined macro. The OMX\_SetupTunnel method will make calls to the components' ComponentTunnelRequest( ) method to set up the tunnel.

By default all ports are created not connected to any other port via a tunnel. After OMX\_SetupTunnel returns successfully, both ports are connected together. Only after calling OMX\_TeardownTunnel are these ports no longer connected.  
OMX\_SetupTunnel shall only be called on ports that are not connected to any other ports.

When changing an input port to non-tunneled communication, the value of the hOutput parameter shall be 0x0. When changing an output port to a non-tunneled communication, the value of the hInput parameter shall be 0x0.

When setting up tunneled communication between an output port and an input port, the method first issues a call to ComponentTunnelRequest( ) on the component with the output port. If the call is successful, a second call to ComponentTunnelRequest( ) on the component with the input port is made.

Should either call to ComponentTunnelRequest( ) fail, the method will set up both the output and input ports for non-tunneled communication.

It is the responsibility of the input port to check that it is compatible with the output port.  
The compatibility check shall be performed:

- At tunnel setup time when the ComponentTunnelRequest( ) method is called on the input port. OMX\_ErrorPortsNotCompatible shall be returned if the compatibility check fails.
- When the input port is enabled. The component shall issue an OMX\_EventError event with the value OMX\_ErrorPortsNotCompatible when the compatibility check fails.
- When the component is transitioning from OMX\_StateLoaded to OMX\_StateIdle state. The component shall issue an OMX\_EventError event with the value OMX\_ErrorPortsNotCompatible when the compatibility check fails.

When checking the compatibility between two ports, the following rules shall be used:

- The domains (eDomain field within the OMX\_PARAM\_PORTDEFINITIONTYPE structure) of the two ports shall be compatible. In certain use cases, OMX\_PortDomainImage and OMX\_PortDomainVideo domains may be considered as compatible.
- When the domain is OMX\_PortDomainAudio, the values of the eEncoding field within the OMX\_AUDIO\_PORTDEFINITIONTYPE structures of the two ports shall either be the same or be equal to OMX\_AUDIO\_CodingUnused.
- When the domain is OMX\_PortDomainVideo or OMX\_PortDomainImage, the values of the nFrameWidth, nFrameHeight, nStride, nSliceHeight, nBitrate, and xFramerate fields of the OMX\_VIDEO\_PORTDEFINITIONTYPE or OMX\_IMAGE\_PORTDEFINITIONTYPE structures of the two ports shall be the same. The eCompressionFormat of both ports shall be the same or eColorFormat of both ports, of the OMX\_VIDEO\_PORTDEFINITIONTYPE or OMX\_IMAGE\_PORTDEFINITIONTYPE structures, shall be the same.
- When the domain is OMX\_PortDomainOther, the values of the eFormat field of the OMX\_OTHER\_FORMATTYPE structures of the two ports shall be the same.
- In case the eEncoding or eCompressionFormat or eFormat or eColorFormat fields of both ports are different from "unused" value (e.g. OMX\_AUDIO\_CodingUnused), then the component shall check the detailed settings of the current format by doing a OMX\_GetParameter on the output port and comparing with the values of the input port.

For example if the ports are of the audio domain and eEncoding is OMX\_AUDIO\_CodingAAC, then the component shall do a OMX\_GetParameter with OMX\_IndexParamAudioAac index to retrieve the detailed AAC settings of the output port and check that they are compatible with the settings of the input port. If OMX\_GetParameter returns OMX\_ErrorUnsupportedIndex or if the values of the fields are either unknown, don't care or variable, the component should assume that the tunneled component does not know the detailed settings, and the compatibility check shall succeed.

The components may negotiate proprietary communication in place of tunneled communication so long as both the output and input ports can support proprietary communication. An IL client cannot disambiguate between tunneled and proprietary communication.

The core should return from this call within 20 milliseconds.

The IL client may use OMX\_SetupTunnel to establish proprietary communication between base profile components (given than both components support it) but not to establish a tunnel between them. An IL client may only establish tunnels between Interop profile components.

If this method fails because the OMX\_SetupTunnel implementation supports neither tunneling nor proprietary communication then it shall return OMX\_ErrorNotImplemented.

If this method fails because OMX\_SetupTunnel supports proprietary communication but not tunneling and proprietary communication does not apply to the given components then it shall return OMX\_ErrorTunnelingUnsupported.

OMX\_SetupTunnel may only return OMX\_ErrorNotImplemented or OMX\_ErrorTunnelingUnsupported when operating on one or more base profile components; these errors do not apply when operating on two Interop profile components.

For a detailed description of the process to set up a data tunnel between two components, see section 3.4.1.2.

OMX\_SetupTunnel is defined as follows.

```
OMX_API OMX_ERRORTYPE OMX_APINTRY OMX_SetupTunnel(
    OMX_IN OMX_HANDLETYPE hOutput,
    OMX_IN OMX_U32 nPortOutput,
    OMX_IN OMX_HANDLETYPE hInput,
    OMX_IN OMX_U32 nPortInput
)
```

The parameters are as follows.

Parameter	Description
<i>hOutput</i> [in]	The handle of the component containing the output port used in the tunnel, where the output port is identified by the <i>nPortOutput</i> parameter. By definition, an output port has the direction OMX_DirOutput. <i>This parameter shall be a valid component handle. If the value of this parameter is 0x0, the hPortInput port on the hInput component will be set up for non-tunneled communication.</i>
<i>nPortOutput</i> [in]	Indicates the output port of the component specified by <i>hOutput</i> that is to be used for tunneled or proprietary communication.
<i>hInput</i> [in]	The handle of the component containing the input port used in the tunnel, where the input port is identified by the <i>nPortInput</i> parameter. By definition, an input port has the direction OMX_DirInput. <i>This parameter shall be a valid component handle. If the value of this parameter is 0x0, the hPortOutput port on the hOutput component will be set up for non-tunneled communication.</i>
<i>nPortInput</i> [in]	Indicates the input port of the component specified by <i>hInput</i> that is to be used for tunneled or proprietary communication.

### 3.2.3.6.1 Prerequisites for This Method

Each component that is being tunneled shall be in the OMX\_StateLoaded state, or its port shall be disabled, *and both ports shall not be currently connected via any current tunnel.*

### 3.2.3.6.2 Results/Outputs for This Method

If the method returns successfully *when both an output and input component are supplied*, tunneled or proprietary communication has been set up between the specified output and input ports, *and both ports are now connected together. If any error is returned then both ports are not connected via a tunnel. When only an output or an input component is supplied or if an error occurs during processing, the ports are set up for non-tunneled communication.*

### 3.2.3.6.3 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* set up tunnel between two components then transition to idle */
OMX_SetupTunnel(hCompA, nCompAOutPort, hCompB, nCompBInPort);
OMX_SendCommand(hCompA, OMX_CommandStateSet, OMX_StateIdle, 0);
OMX_SendCommand(hCompB, OMX_CommandStateSet, OMX_StateIdle, 0);
```

### 3.2.3.7 OMX\_GetContentPipe

The *OMX\_GetContentPipe* method returns a content pipe capable of manipulating a given piece of content as (specified via URI). The OMX IL Core shall provide this interface and return *OMX\_ErrorNotImplemented* if it is not implemented.

The IL client may also use this function to retrieve content pipes for its own use.

The core should return from this call within 20 milliseconds.

OMX\_GetContentPipe is defined as follows.

```
OMX_API OMX_ERRORTYPE OMX_APientry OMX_GetContentPipe(→
→ OMX_IN OMX_HANDLETYPE *hPipe,
→ OMX_IN OMX_STRING szURI →)
```

The parameters are as follows.

**Parameter**

**Description**

hPipe The handle of content pipe retrieved.

{out}

szURI The name of the content the caller is requesting an associated content pipe for.  
{in}

#### 3.2.3.7.1 Prerequisites for This Method

None.

#### 3.2.3.7.2 Results/Outputs for This Method

### 3.2.3.7 The IL Core populates the hPipe field with a content pipe handle corresponding to the given URI. OMX\_TeardownTunnel

The OMX\_TeardownTunnel method clears tunneled communication between an output port and an input port. This method is an actual method and not a defined macro. The OMX\_TeardownTunnel method will make calls to the components' ComponentTunnelRequest( ) methods to tear down the tunnel.

OMX\_TeardownTunnel shall only be called on a pair of ports that are currently connected together. After OMX\_TeardownTunnel returns successfully these ports are no longer connected together.

The core should return from this call within 20 milliseconds.

The IL client may use OMX\_TeardownTunnel to disconnect components currently using proprietary communication.

OMX\_TeardownTunnel is defined as follows.

```
OMX_API OMX_ERRORTYPE OMX_APientry OMX_TeardownTunnel(
→ OMX_IN OMX_HANDLETYPE hOutput,
→ OMX_IN OMX_U32 nPortOutput,
→ OMX_IN OMX_HANDLETYPE hInput,
→ OMX_IN OMX_U32 nPortInput
→ )
```

The parameters are as follows.

<u>Parameter</u>	<u>Description</u>
<i>hOutput</i> [in]	The handle of the component containing the output port used in the tunnel, where the output port is identified by the <i>nPortOutput</i> parameter. By definition, an output port has the direction OMX_DirOutput. This parameter shall be a valid component handle.
<i>nPortOutput</i> [in]	Indicates the output port of the component specified by <i>hOutput</i> that is currently being used for tunneled or proprietary communication.
<i>hInput</i> [in]	The handle of the component containing the input port used in the tunnel, where the input port is identified by the <i>nPortInput</i> parameter. By definition, an input port has the direction OMX_DirInput. This parameter shall be a valid component handle.
<i>nPortInput</i> [in]	Indicates the input port of the component specified by <i>hInput</i> that is currently being used for tunneled or proprietary communication.

### **3.2.3.7.1 Prerequisites for This Method**

Each component that is being tunneled shall be in the OMX\_StateLoaded state, or its port shall be disabled, and both ports shall be currently connected together.

### **3.2.3.7.2 Results/Outputs for This Method**

If the method returns successfully both ports are no longer connected, either via a tunnel or using proprietary communication. If the method returns unsuccessfully then both ports retain their previous connection status.

### **3.2.3.7.3 Sample Code Showing Calling Sequence**

The following sample code shows the calling sequence.

```
/* tear down a tunnel between two components */
OMX_TeardownTunnel(hCompA, nCompAOutPort, hCompB, nCompBInPort);
```

### **3.2.3.8 OMX\_GetCoreInterface**

The OMX\_GetCoreInterface method returns a new core extension interface by an extension name. This method will allocate the actual extension interface structure, ensures it is populated correctly, and updates the value of \*ppItf with a pointer to the newly created interface. The core should return from this call within 20 milliseconds.

Since interfaces are requested by name, a naming convention is defined. OpenMAX IL interface names are zero terminated strings with the following format:

“OMX.<vendor\_name>.<vendor\_specified\_convention>”.

Vendor name "Khronos" is reserved for interfaces defined by Khronos.

OMX\_GetCoreInterface is defined as follows.

```
OMX_API OMX_ERRORTYPE OMX_APientry OMX_GetCoreInterface (
    OMX_OUT void ** ppItf,
    OMX_IN OMX_STRING cExtensionName)
```

The parameters are as follows.

<u>Parameter</u>	<u>Description</u>
<u>cExtensionName [in]</u>	An OMX_STRING for the name of the extension interface to be instantiated by the IL core. Extension names are strings limited to a length up to 127 characters plus the trailing null for a maximum length of 128 characters.
<u>ppItf [out]</u>	Pointer to an interface pointer to be filled in by the IL core. The caller has to cast the value to the interface type specific to the named extension. In case this method returns an error, the pointer value is unspecified.

### 3.2.3.8.1 Prerequisites for this Method

The OpenMAX IL core shall be initialized.

### 3.2.3.8.2 Results/Outputs for this Method

If successful, the function returns a valid interface pointer to the IL client. The method shall return OMX\_ErrorNotImplemented if the extension requested by the IL client is not supported.

### 3.2.3.9 OMX\_FreeCoreInterface

The OMX\_FreeCoreInterface method will free an interface allocated by the OMX\_GetCoreInterface method. The exact state the interface needs to be in, when this method is called, is extension-dependent and shall be documented in extension documentation. The core should return from this call within 20 milliseconds.

OMX\_FreeCoreInterface is defined as follows.

```
OMX_API OMX_ERRORTYPE OMX_APientry OMX_FreeCoreInterface (
    OMX_IN void * pItf)
```

The single parameter is defined as follows.

<u>Parameter</u>	<u>Description</u>
<u>pItf [in]</u>	The pointer to the interface to be freed.

### 3.2.3.9.1 Prerequisites for this Method

The input parameter pItf shall contain a valid pointer to an interface allocated by the OMX\_GetCoreInterface method.

### **3.2.3.9.2 Results/Outputs for this Method**

The method has freed the interface.

## **3.3 OpenMAX IL Component Methods and Structures**

OpenMAX IL components are defined in the OMX\_Component.h header file. The structure OMX\_COMPONENTTYPE holds the data fields and function entry points for a component.

OMX\_COMPONENTTYPE is defined as follows.

```
typedef struct OMX_COMPONENTTYPE
{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_PTR pComponentPrivate;
    OMX_PTR pApplicationPrivate;

    OMX_ERRORTYPE (*GetComponentVersion)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_OUT OMX_STRING pComponentName,
        OMX_OUT OMX_VERSIONTYPE* pComponentVersion,
        OMX_OUT OMX_VERSIONTYPE* pSpecVersion,
        OMX_OUT OMX_UUIDTYPE* pComponentUUID);

    OMX_ERRORTYPE (*SendCommand)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_COMMANDTYPE Cmd,
        OMX_IN OMX_U32 nParam1,
        OMX_IN OMX_PTR pCmdData);

    OMX_ERRORTYPE (*GetParameter)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_INDEXTYPE nIndex,
        OMX_INOUT OMX_PTR pComponentParameterStructure);

    OMX_ERRORTYPE (*SetParameter)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_INDEXTYPE nIndex,
        OMX_IN OMX_PTR pComponentParameterStructure);

    OMX_ERRORTYPE (*GetConfig)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_INDEXTYPE nIndex,
        OMX_INOUT OMX_PTR pComponentConfigStructure);

    OMX_ERRORTYPE (*SetConfig)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_INDEXTYPE nIndex,
        OMX_IN OMX_PTR pComponentConfigStructure);

    OMX_ERRORTYPE (*GetExtensionIndex)(
        OMX_IN OMX_HANDLETYPE hComponent,
        OMX_IN OMX_STRING cParameterName,
```

```

    OMX_OUT OMX_INDEXTYPE* pIndexType);

OMX_ERRORTYPE (*GetState)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_OUT OMX_STATETYPE* pState);

OMX_ERRORTYPE (*ComponentTunnelRequest)(
    OMX_IN OMX_HANDLETYPE hComp,
    OMX_IN OMX_U32 nPort,
    OMX_IN OMX_HANDLETYPE hTunneledComp,
    OMX_IN OMX_U32 nTunneledPort,
    OMX_INOUT OMX_TUNNELSETUPTYPE* pTunnelSetup);

OMX_ERRORTYPE (*UseBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_INOUT OMX_BUFFERHEADERTYPE** ppBufferHdr,
    OMX_IN OMX_U32 nPortIndex,
    OMX_IN OMX_PTR pAppPrivate,
    OMX_IN OMX_U32 nSizeBytes,
    OMX_IN OMX_U8* pBuffer);

OMX_ERRORTYPE (*AllocateBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_INOUT OMX_BUFFERHEADERTYPE** ppBuffer,
    OMX_IN OMX_U32 nPortIndex,
    OMX_IN OMX_PTR pAppPrivate,
    OMX_IN OMX_U32 nSizeBytes);

OMX_ERRORTYPE (*FreeBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_IN OMX_U32 nPortIndex,
    OMX_IN OMX_BUFFERHEADERTYPE* pBuffer);

OMX_ERRORTYPE (*EmptyThisBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_IN OMX_BUFFERHEADERTYPE* pBuffer);

OMX_ERRORTYPE (*FillThisBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_IN OMX_BUFFERHEADERTYPE* pBuffer);

OMX_ERRORTYPE (*SetCallbacks)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_IN OMX_CALLBACKTYPE* pCallbacks,
    OMX_IN OMX_PTR pAppData);

OMX_ERRORTYPE (*ComponentDeInit)(
    OMX_IN OMX_HANDLETYPE hComponent);

OMX_ERRORTYPE (*UseEGLImage)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_INOUT OMX_BUFFERHEADERTYPE** ppBufferHdr,
    OMX_IN OMX_U32 nPortIndex,
    OMX_IN OMX_PTR pAppPrivate,
    OMX_IN void* eglImage);

OMX_ERRORTYPE (*ComponentRoleEnum)(
```

```

    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_OUT OMX_U8 *cRole,
    OMX_IN OMX_U32 nIndex);
}

} OMX_COMPONENTTYPE;

```

### **3.3.1 *pComponentPrivate***

*pComponentPrivate* is a pointer to the component private data area. The component allocates and initializes this member when the component is first loaded. The application should not access this data area.

### **3.3.2 *pApplicationPrivate***

*pApplicationPrivate* is a pointer to the application private data area. The component initializes this field during the call to *SetCallbacks*, as this field is provided back to the IL client when the component issues callbacks.

### **3.3.3 *GetComponentVersion***

The IL client calls the *GetComponentVersion* component method via the *OMX\_GetComponentVersion* core macro. See the definition of *OMX\_GetComponentVersion* in section 3.2.2.1 above for a description of its semantics.

*GetComponentVersion* is defined as follows:

```

OMX_ERRORTYPE (*GetComponentVersion)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_OUT OMX_STRING pComponentName,
    OMX_OUT OMX_VERSIONTYPE* pComponentVersion,
    OMX_OUT OMX_VERSIONTYPE* pSpecVersion);

```

### **3.3.4 *SendCommand***

The IL client calls the *SendCommand* component method via the *OMX\_SendCommand* core macro. See the definition of *OMX\_SendCommand* in section 3.2.2.2 above for a description of its semantics.

*SendCommand* is defined as follows:

```

OMX_ERRORTYPE (*SendCommand)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_IN OMX_COMMANDTYPE Cmd,
    OMX_IN OMX_U32 nParam,
    OMX_IN OMX_PTR pCmdData);

```

### **3.3.5    *GetParameter***

The IL client or a tunneled component calls the *GetParameter* component method via the OMX\_GetParameter core macro. See the definition of OMX\_GetParameter in section 3.2.2.8 above for a description of its semantics.

*GetParameter* is defined as follows:

```
OMX_ERRORTYPE (*GetParameter)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_INDEXTYPE nParamIndex,  
    OMX_INOUT OMX_PTR pComponentParameterStructure);
```

### **3.3.6    *SetParameter***

The IL client or a tunneled component calls the *SetParameter* component method via the OMX\_SetParameter core macro. See the definition of OMX\_SetParameter in section 3.2.2.8.3 above for a description of its semantics.

*SetParameter* is defined as follows:

```
OMX_ERRORTYPE (*SetParameter)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_INDEXTYPE nIndex,  
    OMX_IN OMX_PTR pComponentParameterStructure);
```

### **3.3.7    *GetConfig***

The IL client calls the *GetConfig* component method via the OMX\_GetConfig core macro. See the definition of OMX\_GetConfig in section 3.2.2.9.3 above for a description of its semantics.

*GetConfig* is defined as follows:

```
OMX_ERRORTYPE (*GetConfig)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_INDEXTYPE nIndex,  
    OMX_INOUT OMX_PTR pComponentConfigStructure);
```

### **3.3.8    *SetConfig***

The IL client calls the *SetConfig* component method via the OMX\_SetConfig core macro. See the definition of OMX\_SetConfig in section 3.2.2.10.3 above for a description of its semantics.

*SetConfig* is defined as follows:

```
OMX_ERRORTYPE (*SetConfig)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_INDEXTYPE nIndex,  
    OMX_IN OMX_PTR pComponentConfigStructure);
```

### **3.3.9    *GetExtensionIndex***

The IL client calls the *GetExtensionIndex* component method via the `OMX_GetExtensionIndex` core macro. See the definition of `OMX_GetExtensionIndex` in section 3.2.2.12 for a description of its semantics.

*GetExtensionIndex* is defined as follows:

```
OMX_ERRORTYPE (*GetExtensionIndex)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_STRING eParameterName,  
    OMX_OUT OMX_INDEXTYPE* pIndexType);
```

### **3.3.10    *GetState***

The IL client calls the *GetState* component method via the `OMX_GetState` core macro. See the definition of `OMX_GetState` in section 3.2.2.13 above for a description of its semantics.

*GetState* is defined as follows:

```
OMX_ERRORTYPE (*GetState)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_OUT OMX_STATETYPE* pState);
```

### **3.3.11    *ComponentTunnelRequest***

The *ComponentTunnelRequest* method will interact with another OpenMAX IL component to determine if tunneling is possible and to set up the tunneling if it is possible. The return codes for this method can determine if tunneling is not possible or if proprietary communication or tunneling is used. [The ComponentTunnelRequest method will also be used to disconnect a port that was previously connected via tunneling or proprietary communication.](#)

The interop profile-conformant component shall support tunneling to a component with compatible parameters ([refer to 3.2.3.6 for information regarding compatibility checking](#)). The component may also support proprietary communication. If proprietary communication is supported, the negotiation of proprietary communication is performed in a vendor-specific way. The only requirement is that the proper result be returned. The details of the proprietary communication setup are left to the vendor's component implementer.

The *ComponentTunnelRequest* method is invoked on both components that support the tunneling communication. When this method is invoked on the component that provides the output port, the component will do the following:

1. Indicate its supplier preference in `pTunnelSetup`.

When this method is invoked on the component that provides the input port, the component will do the following:

1. Check the data compatibility between the ports using one or more `GetParameter` calls.
2. Review the buffer supplier preferences of the output port and use `OMX_SetParameter` with index `OMX_IndexParamCompBufferSupplier` to inform the output port of which port supplies the buffers.

If this method is invoked with a NULL `valueparameter` for the `pTunnelComp` parameter, the port should be set up for non-tunneled communication with the IL client, and is no longer connected to another port.

If this method is invoked with a non-NULL value for the `pTunnelComp` parameter, and this port is already connected then this call shall fail.

The component should return from this call within five milliseconds.

`ComponentTunnelRequest` is defined as follows:

```
OMX_ERRORTYPE (*ComponentTunnelRequest)(
    OMX_IN OMX_HANDLETYPE hComp,
    OMX_IN OMX_U32 nPort,
    OMX_IN OMX_HANDLETYPE hTunneledComp,
    OMX_IN OMX_U32 nTunneledPort,
    OMX_INOUT OMX_TUNNELSETUPTYPE* pTunnelSetup);
```

The parameters for this method are defined as follows.

Parameter	Description
<code>hComp</code> [in]	The handle of the target component of the <code>RequestTunnel</code> call and one of the components that will participate in the tunnel.
<code>nPort</code> [in]	The index of the port belonging to <code>hComp</code> that will participate in the tunnel.
<code>hTunneledComp</code> [in]	The handle of the other component that participates in the tunnel. When this parameter is NULL, the port specified in <code>nPort</code> should be configured for non-tunneled communication with the IL client <u>and not connected to another port</u> .
<code>nTunneledPort</code> [in]	The index of the port belonging to <code>hTunneledComp</code> that participates in the tunnel.
<code>pTunnelSetup</code> [in,out]	The structure that contains data for the tunneling negotiation between components. The supplier field can be filled by both components; <u>the callbacks field is filled by the output port component</u> . The read-only flag can be applied by both components.

### 3.3.11.1 Prerequisites for This Method

The component shall be in the `OMX_StateLoaded` state or its port shall be disabled.

### 3.3.11.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* Translate a SetupTunnel call to two ComponentTunnelRequest calls */
pCompOut = (OMX_COMPONENTTYPE *)hOutput;
pCompIn = (OMX_COMPONENTTYPE *)hInput;
pCompOut->ComponentTunnelRequest(hOutput, nPortOutput, hInput,
    nPortInput, &oTunnelSetup);
pCompIn->ComponentTunnelRequest(hInput, nPortInput, hOutput,
    nPortOutput, &oTunnelSetup);
```

### 3.3.12 UseBuffer

The IL client or a tunneled component calls the UseBuffer component method via the OMX\_UseBuffer core macro. See the definition of OMX\_UseBuffer in section 3.2.2.14 above for a description of its semantics.

**UseBuffer** is defined as follows:

```
OMX_ERRORTYPE (*UseBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_INOUT OMX_BUFFERHEADERTYPE** ppBufferHdr,
    OMX_IN OMX_U32 nPortIndex,
    OMX_IN OMX_PTR pAppPrivate,
    OMX_IN OMX_U32 nSizeBytes,
    OMX_IN OMX_U8* pBuffer);
```

### 3.3.13 AllocateBuffer

The IL client calls the AllocateBuffer component method via the OMX\_AllocateBuffer core macro. See the definition of OMX\_AllocateBuffer in section 3.2.2.15 above for a description of its semantics.

**AllocateBuffer** is defined as follows:

```
OMX_ERRORTYPE (*AllocateBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_INOUT OMX_BUFFERHEADERTYPE** pBuffer,
    OMX_IN OMX_U32 nPortIndex,
    OMX_IN OMX_PTR pAppPrivate,
    OMX_IN OMX_U32 nSizeBytes);
```

### 3.3.14 FreeBuffer

The IL client or a tunneled component calls the FreeBuffer component method via the OMX\_FreeBuffer core macro. See the definition of OMX\_FreeBuffer in section 3.2.2.16 above for a description of its semantics.

**FreeBuffer** is defined as follows:

```
OMX_ERRORTYPE (*FreeBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_IN OMX_U32 nPortIndex,
```

```
OMX_IN OMX_BUFFERHEADERTYPE* pBuffer);
```

### 3.3.15 EmptyThisBuffer

The IL client or a tunneled component calls the `EmptyThisBuffer` component method via the `OMX_EmptyThisBuffer` core macro. See the definition of `OMX_EmptyThisBuffer` in section 3.2.2.17 above for a description of its semantics.

`EmptyThisBuffer` is defined as follows:

```
OMX_ERRORTYPE (*EmptyThisBuffer)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_BUFFERHEADERTYPE* pBuffer);
```

### 3.3.16 FillThisBuffer

The IL client or a tunneled component calls the `FillThisBuffer` component method via the `OMX_FillThisBuffer` core macro. See the definition of `OMX_FillThisBuffer` in section 3.2.2.18 above for a description of its semantics.

`FillThisBuffer` is defined as follows:

```
OMX_ERRORTYPE (*FillThisBuffer)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_BUFFERHEADERTYPE* pBuffer);
```

### 3.3.17 SetCallbacks

The IL client calls the `SetCallbacks` component method via the `OMX_SetCallbacks` core macro. The `SetCallbacks` method will allow the core to transfer the callback structure from the IL client to the component. This is a blocking call. ~~The component should return from this call within five milliseconds. See the definition of `OMX_SetCallbacks` in section 3.2.2.20 for a description of its semantics.~~

`SetCallbacks` is defined as follows:

```
OMX_ERRORTYPE (*SetCallbacks)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_CALLBACKTYPE* pCallbacks,  
    OMX_IN OMX_PTR pAppData);
```

The parameters are as follows:

Parameter	Description
<code>hComponent</code>	The handle of the component that executes the call. [in]
<code>pCallbacks</code>	A pointer to an <code>OMX_CALLBACKTYPE</code> structure that is used to provide the callback information to the component. [in]

Parameter	Description
<i>pAppData</i> [in]	A pointer to a value that the IL client has defined (for example, a pointer to a data structure) that allows the callback in the IL client to determine the context of the call.

### 3.3.17.1 Prerequisites for This Method

The component shall be in the OMX\_StateLoaded state.

### 3.3.17.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```
/* On GetHandle (for statically linked components):
   create component, initialize it, and set its callbacks */
pComp = (OMX_COMPONENTTYPE *)malloc(sizeof(OMX_COMPONENTTYPE));
hHandle = (OMX_HANDLETYPE)pComp;
pComp->nVersion = version_1_2_0;
pComp->nSize = sizeof(OMX_COMPONENTTYPE);
OMX_ComponentRegistered(hHandle, pInitialize, pAppData);
pComp->SetCallbacks(hHandle, pCallBacks, pAppData);
```

### 3.3.18 ComponentDeinit

The core calls the ComponentDeinit function when the core needs to dispose of a component.

ComponentDeinit is defined as follows.

```
OMX_ERRORTYPE (*ComponentDeinit)(
    OMX_IN OMX_HANDLETYPE hComponent);
```

The single parameter for this method is as follows.

#### Parameter Description

<i>hComponent</i> [in]	The handle of the component that executes the call.
---------------------------	---

### 3.3.18.1 Prerequisites for This Method

The IL client shall call this method the OMX\_FreeHandle macro only when the component is in OMX\_StateLoaded state. There are no prerequisites for this method. The IL client may execute this function regardless of component state so that de-initialization is guaranteed even on components that are unresponsive to state changes. However, executing ComponentDeinit when the component is in the OMX\_StateLoaded state is recommended for proper shutdown..

### 3.3.18.2 Sample Code Showing Calling Sequence

The following sample code shows the calling sequence.

```

/* On FreeHandle: de-initialize component and destroy it */
pComp = (OMX_COMPONENTTYPE*)hComponent;
(pComp->ComponentDeinit)(hComponent);
OMX_OSAL_Free(pComp);

```

### 3.3.19 UseEGLImage

The IL client or a tunneled component calls the UseEGLImage component method via the OMX\_UseEGLImageBuffer core macro. See the definition of OMX\_UseEGLImage in section 3.2.2.19 above for a description of its semantics.

UseEGLImageBuffer is defined as follows.

```

OMX_ERRORTYPE (*UseEGLImageBuffer)(
    OMX_IN OMX_HANDLETYPE hComponent,
    OMX_INOUT OMX_BUFFERHEADERTYPE** ppBufferHdr,
    OMX_IN OMX_U32 nPortIndex,
    OMX_IN OMX_PTR pAppPrivate,
    OMX_IN void* pBuffer);

```

## 3.4 Calling Sequences

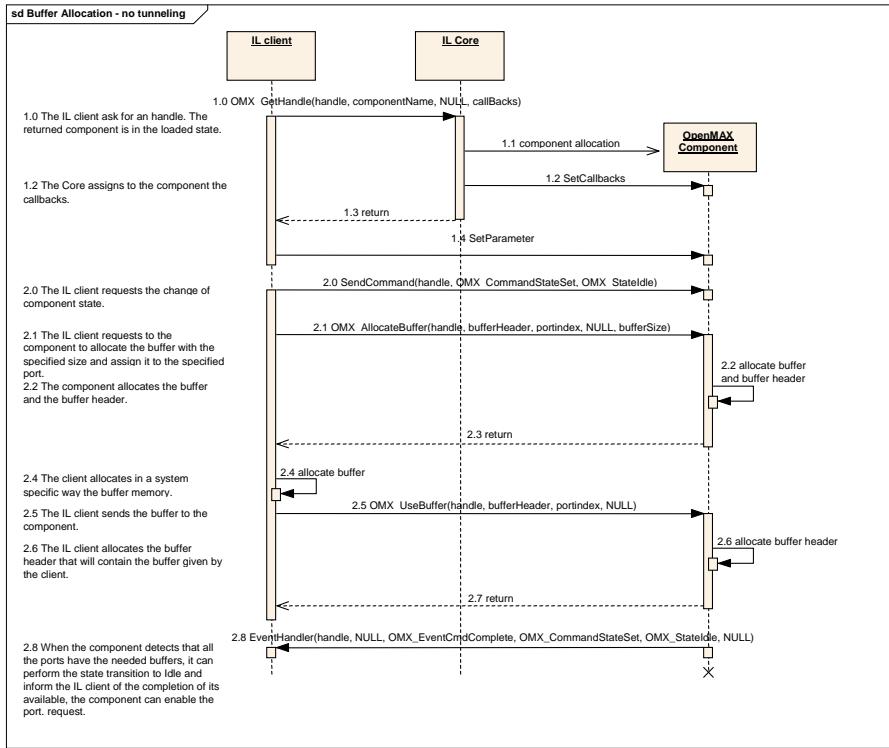
This section describes how the IL client, the OpenMAX IL core, and the components dynamically interact in a few meaningful use cases, namely initialization, de-initialization, data flow, data tunneling setup, and data flow in the case of data tunneling and dynamic port reconfiguration. The interaction between the core, the components, and the possible implementation of a resource manager is also described.

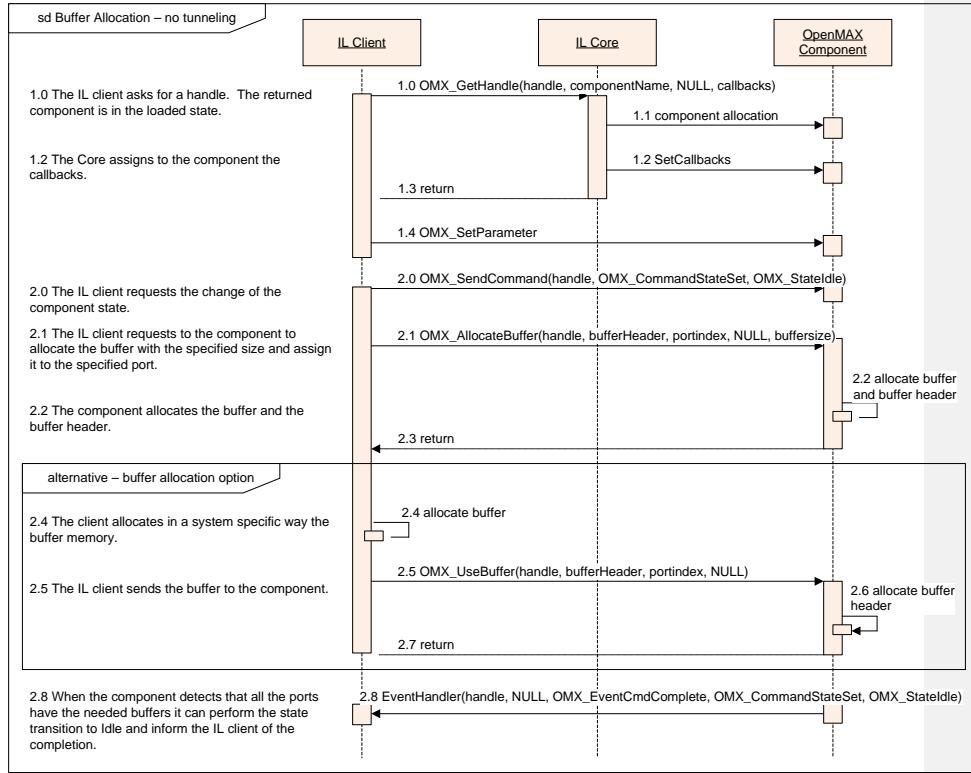
### 3.4.1 Initialization

This section describes the operations for initializing the OpenMAX IL components. The components can be handled directly by the IL client, can be tunneled to each other, or both. The tunneled and non-tunneled cases are distinguished for clarity, but the two cases can be both present in the component framework.

#### 3.4.1.1 Non-tunneled Initialization

Figure 3-9 shows how an IL client should initialize an OpenMAX IL component.





**Figure 3-9. Component Initialization**

First, the IL client shall call the `OMX_GetHandle` function, which activates the actual component creation (1.1) by the core. Also, all of the configuration resources of the component are loaded into memory. The core passes IL client callback functions to the component by means of the `SetCallbacks` method (1.2). If previous steps are successful, a valid handle is returned in step 1.3 and the component will be in the `OMX_StateLoaded` state.

The IL client shall configure the component and its ports. For this purpose, the IL core macro `OMX_SetParameter` shall be used; it may be called multiple times (step 1.4) if needed.

When the client has completed the configuration phase, it can request the component to make the state transition to `OMX_StateIdle`. Only after this request shall the IL client set up buffers for the component to use for all of its ports. The IL client shall use either `OMX_AllocateBuffer` or `OMX_UseBuffer` to set up buffers. If the IL client asks components for a tunnel, it does not allocate **setup**-buffers because the tunneled components allocate any buffers. See section 3.4.1.2 for more details on tunneling.

This process may be repeated multiple times, depending on the number of ports and the total number of buffers needed on each port. If OMX\_UseBuffer is used, the IL client may pre-announce an allocated buffer to the component, or may defer the allocation and announce a NULL buffer pointer shall have allocated a buffer and passed it to the component. Alternatively, the IL client may ask the component to allocate a buffer and a buffer header using the OMX\_AllocateBuffer method. In the latter case, the component will allocate both a buffer and its related header and return it to the IL client by reference.

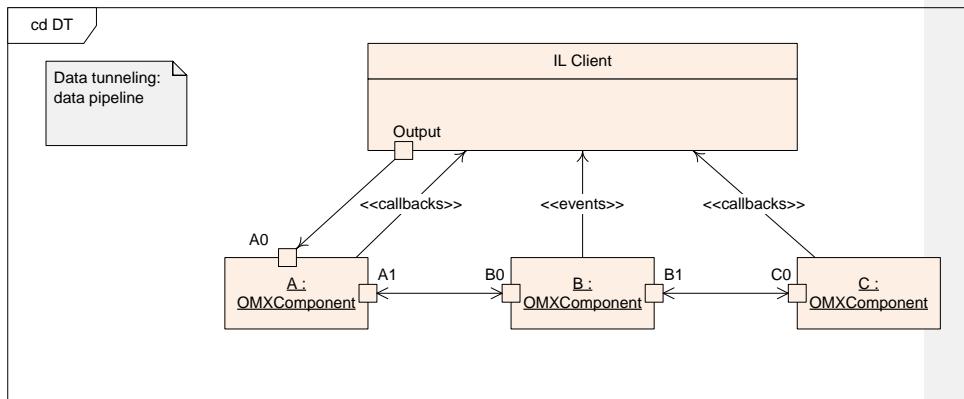
As soon as these initial configuration steps are completed, the component shall complete the state transition and return an event to the client for the SendCommand request completion (step 2.8).

The component is now ready to be used by the IL client.

### 3.4.1.2 Tunneled Initialization

To avoid moving data buffers back and forth among the IL client and OpenMAX IL components, data tunnels can be set up so that the output buffer of one component is passed directly to the input port of the next component in the chain.

Consider the example shown in [Figure 3-10](#), where an IL client generates data for a chain of three tunneled components identified as A, B, and C. Component C is a sink and does not return data to the IL client.



**Figure 3-10. Example of Data Tunneling Among OpenMAX IL Components**

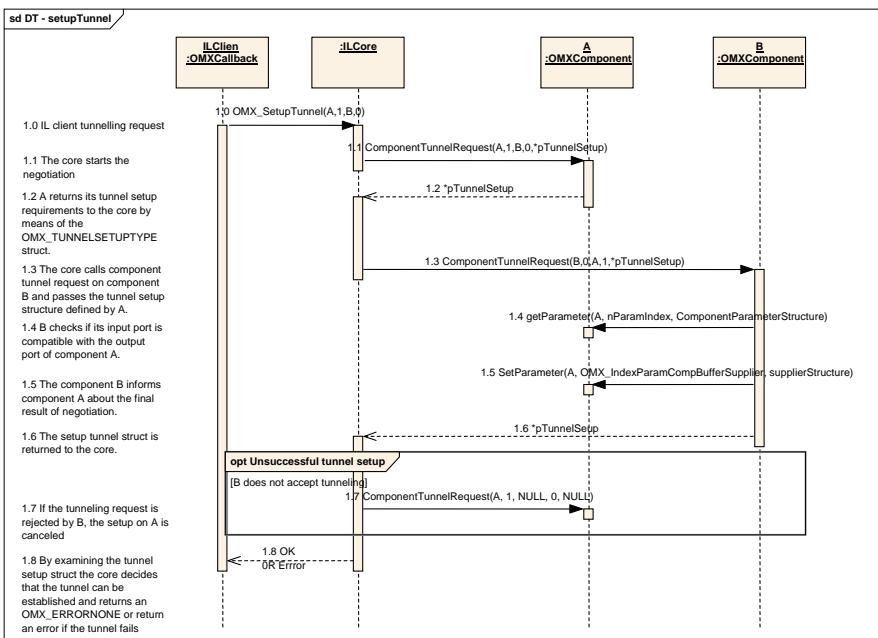
Note that all callbacks are always directed to and managed by the IL client when ports communicate using proprietary or tunneled communication. The tunneling setup and initialization require a detailed description, based on the following steps:

- The components are constructed with the calls to OMX\_GetHandle.

- The components are tunneled, linking an output port of the first component to an input port of the second component. The port that shall supply the buffer is decided in this phase.
- The IL client may override the **input** ports' choice of buffer supplier after OMX\_SetupTunnel has completed by setting the buffer supplier into the input port, which in turn will reprogram the supplier to the output port.

During the transition from OMX\_StateLoaded to OMX\_StateIdle, each component shall not transition until the required buffer **headers** on all enabled ports have been allocated.

OMX\_SetupTunnel shall be executed only when the components are in the OMX\_StateLoaded state or when ports are disabled, and when those ports are not connected to any others. [Figure 3-11](#) illustrates the setup process:



[Figure 3-11. Tunnel Setup](#)

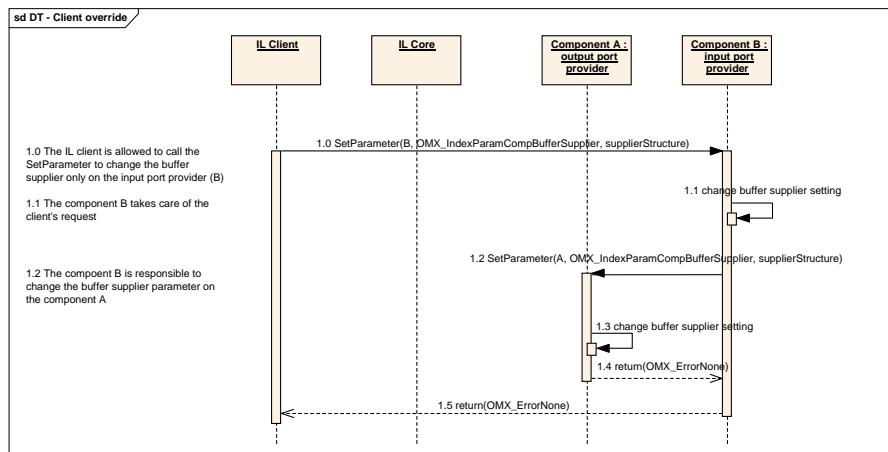
The IL client shall start the data setup process by calling the OMX\_SetupTunnel function of the IL core when the components that are being tunneled are in the OMX\_StateLoaded state (step 1.0).

As a result, the IL core shall call the ComponentTunnelRequest methods of component A and B in sequence. The structure OMX\_TUNNELSETUPTYPE defined in section 3.1.3.10 shall be passed by the IL core to the component with the output port first. The component receiving such a call shall fill in the structure and return it to the core. If the ComponentTunnelRequest call returns successfully, the IL core shall call the

same function on the second component (1.3), passing the OMX\_TUNNELSETUPTYPE structure that was filled in by the first component. The component also shall check that the output port of the peer component is compatible with its input port (i.e., the data type should be the same) (1.4). If the tunnel setup parameters included in the structure are agreed to by the second component, the ComponentTunnelRequest call will send back to the first component the result of negotiation (1.5) and returns successfully (1.6). The IL core shall check that both calls of ComponentTunnelRequest did not return errors. If so, the initial OMX\_SetupTunnel will return successfully.

If the call to ComponentTunnelRequest on component B fails, component A will be set to not tunnel by a second call to ComponentTunnelRequest with a pointer to NULL in place of the component B handle and pTunnelSetup parameter.

After the successful tunnel setup, the IL client may override the buffer supplier negotiation with the procedure illustrated in [Figure 3-12](#):

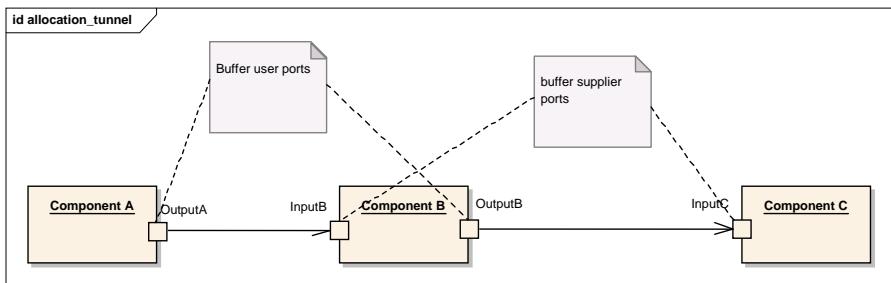


**Figure 3-12. IL Client Buffer Supplier Override**

If the IL client wants to override the negotiation of tunneled components that specifies which component is the buffer supplier, it shall call the function SetParameter on the component that provides the input port. That component is responsible for signaling to the other tunneled component the new buffer supplier, with the same call to SetParameter.

Note that if the IL client calls SetParameter before the tunnel setup, an IL component shall return OMX\_ErrorIncorrectStateOperation as changing the buffer supplier preference before a tunnel is actually setup is not supported.

The last step of the tunnel initialization phase is the state transition from OMX\_StateLoaded to OMX\_StateIdle that also involves the buffer allocation and assignment. [Figure 3-13](#) illustrates the state transition behavior in which the tunnels are already created and configured.

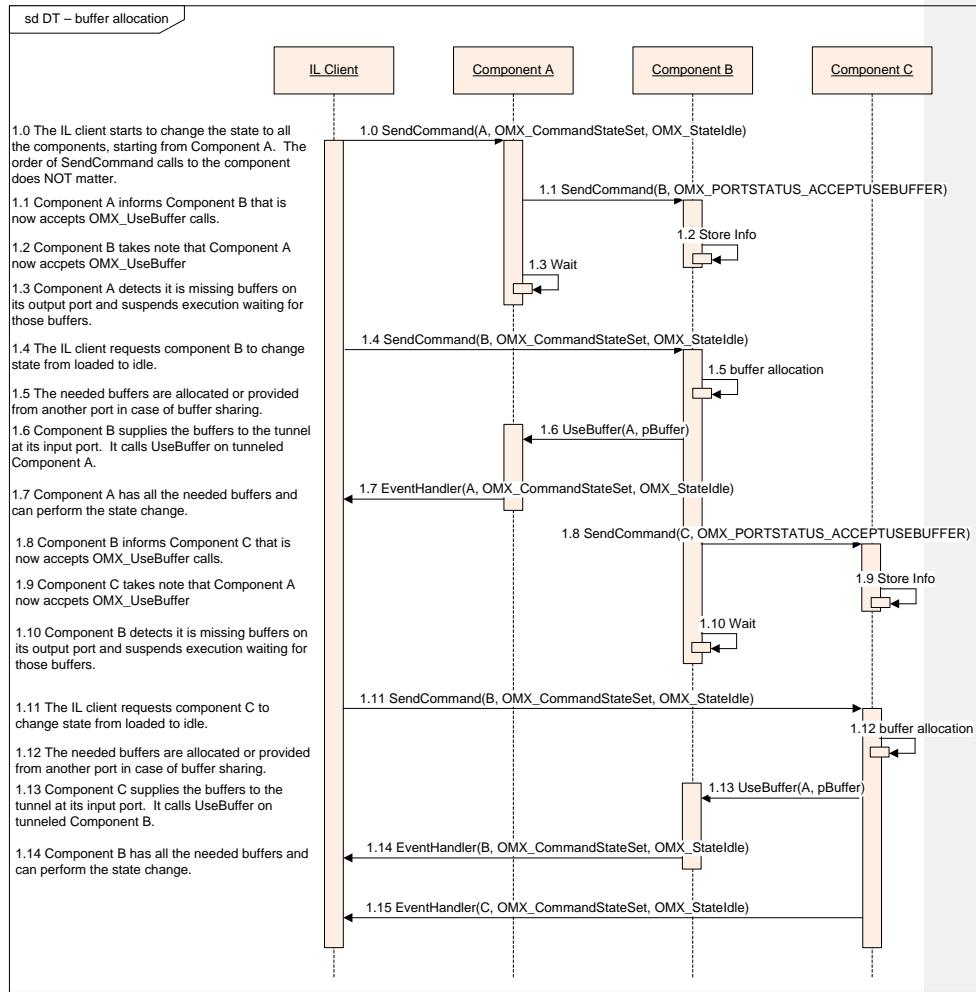


**Figure 3-13. Tunneling Example**

Component A is tunneled with component B, and component B is the buffer supplier. Component B is tunneled with component C, and component C is the buffer supplier.

[Figure 3-14](#) illustrates the behavior of each tunneled component during the state transition.





**Figure 3-14. State Transition to Idle in the Case of Tunneled Components**

Each supplier port on a component shall pass its buffers or NULL pointers to the non-supplier port it is tunneling with via `OMX_UseBuffer`. After all of its supplier ports have passed buffers, the component waits until all of its non-supplier ports have received all of their buffers via `OMX_UseBuffer`.

In [Figure 3-14](#), component A receives the state transition request from the IL client. Component A is tunneled with component B. The input port of B is set as buffer supplier for the tunnel. In this case, component A shall wait until its output port receives all of the needed OMX\_UseBuffer callsbuffers.

Meanwhile, the IL client asks component B to change its state. In this case, component B has a port that is a buffer supplier, the input port, and it shall call OMX\_UseBuffer on the output port of component A to allocate the buffer headers necessary for data transfer. Component B may allocate the buffers at this point and can pre-announce them, or may defer buffer allocation and will use NULL buffer pointers. Then, component B waits for all of the OMX\_UseBuffer calls needed buffers on its output port.

Now component A has allocated all of the needed buffer headers, so it can perform the state transition to OMX\_StateIdle. The exact sequence of transitions can be different, since it depends on the platform, the operating system, and the implementation. The only rule is to wait until all the resources are available.

The IL client requests that component C change its state. Component C behaves like component B: Component C makes the necessary OMX\_UseBuffer calls on the output port of gives the buffers needed to component B, and then can change its state, since it does not need any other buffers.

Finally, component B can change its state to OMX\_StateIdle since it has allocated obtained all of the needed buffer headers.

### 3.4.2 Data Flow

OpenMAX IL defines two means of data communication:

- Tunneled communication, where a port exchanges data directly with a port on another component
- Non-tunneled communication, where a port exchanges data only with the IL client

A port may implement data tunneling via proprietary communication, taking advantage of platform-specific features. The following sections describe the data flow inherent to each means of communication.

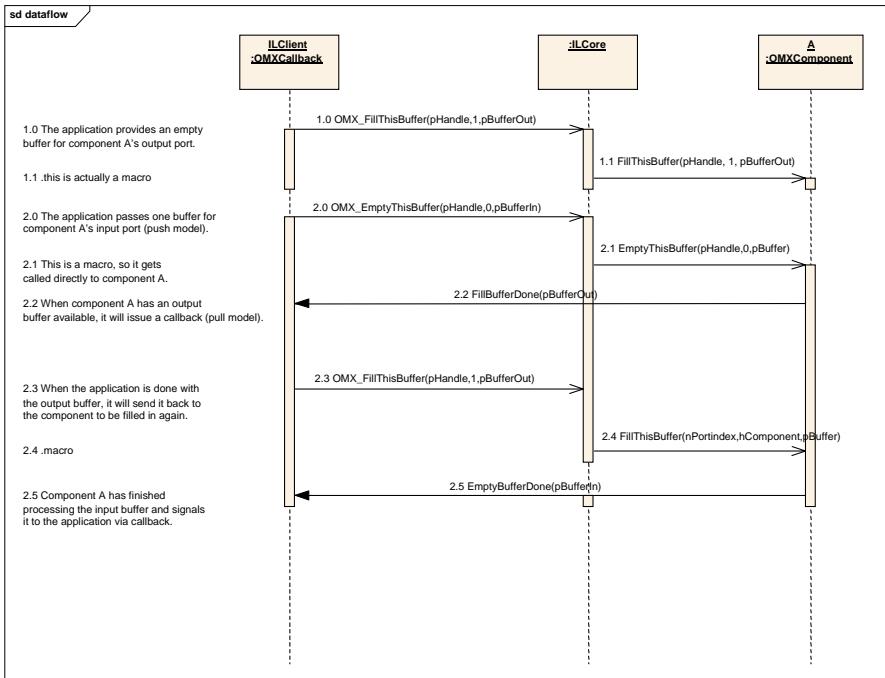
#### 3.4.2.1 Non-tunneled Data Flow

An IL client that has a data buffer to deliver to a component input port shall issue an OMX\_EmptyThisBuffer call.

Conversely, for the component output port, the IL client shall initially provide one or more empty buffers into which the component can write output data; the OMX\_FillThisBuffer call accomplishes this task. As soon as one buffer is available from the component output port, the component shall send an FillBufferDone callback. The component is aware of the callback entry point from the earlier SetBacks call.

Note that the IL client is entirely responsible for moving data buffers among components if data tunneling is not used.

Figure 3-15 illustrates the dynamic behavior related to data flow.



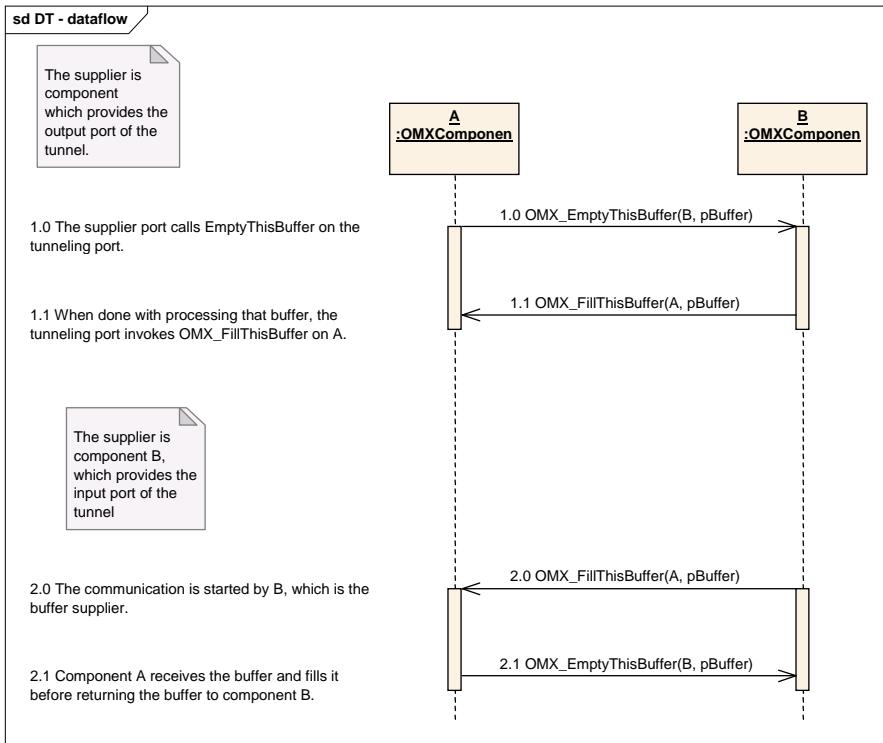
**Figure 3-15. Data Flow Between Non-tunneled Components**

### 3.4.2.2 Tunneled Data Flow

In data tunneling, OpenMAX IL components directly pass data buffers among themselves without returning them to the IL client. This data flow uses a different convention from the situation where all data buffers are exchanged with the IL client.

If the buffer supplier is the output component, it shall call `OMX_EmptyThisBuffer` on the other tunneled component to pass the buffer that is to be emptied. When the input component has terminated the operation, it shall return the buffer to the output component by calling `OMX_FillThisBuffer` on it.

If the buffer supplier is the input component, the communication mechanism is the same but is initiated by calling `OMX_FillThisBuffer` on the output component. [Figure 3-16](#) illustrates this process.

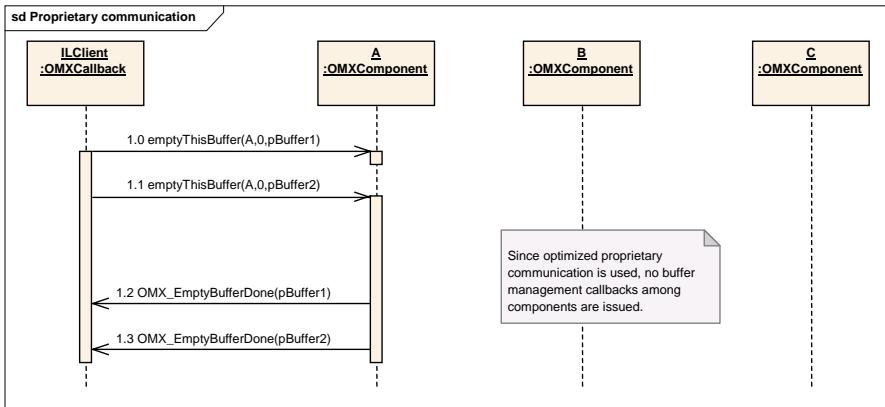


**Figure 3-16. Data Flow Between Tunneled Components**

### 3.4.2.3 Proprietary Communication

On some platforms data tunneling among components can be optimized by proprietary communication mechanisms, which can be based on specific hardware such as DMA or shared memory. Such resources are set up in a proprietary manner during the standard data tunneling setup phase. Although the IL client uses the standard OMX\_SetupTunnel call, platform-specific optimizations can prepare optimized transport channels among components.

Assuming a chain of components A, B, and C that support proprietary communication, the resulting data flow would appear as illustrated in [Figure 3-17](#).



**Figure 3-17. Data Flow with Proprietary Communication Between Components**

Assuming that all components are in the `OMX_StateExecuting` state, the IL client sends two buffers to component A using the `OMX_EmptyThisBuffer` call (steps 1.0 and 1.1). Given the data tunnel setup, the output of component A is sent to the input port of component B. The output of component B is sent to the input port of component C, which is the sink.

No callbacks will be invoked since the components will use their proprietary mechanisms to move data.

The `EmptyBufferDone` callback will be issued to the IL client only when component A has finished processing buffers.

Even though buffer-related callbacks are not used in this use case, note that components may still generate events to the IL client using the `EventHandler` callback entry point.

### 3.4.3 De-Initialization

This section describes tunneled and non-tunneled component de-initialization.

#### 3.4.3.1 Non-tunneled De-initialization

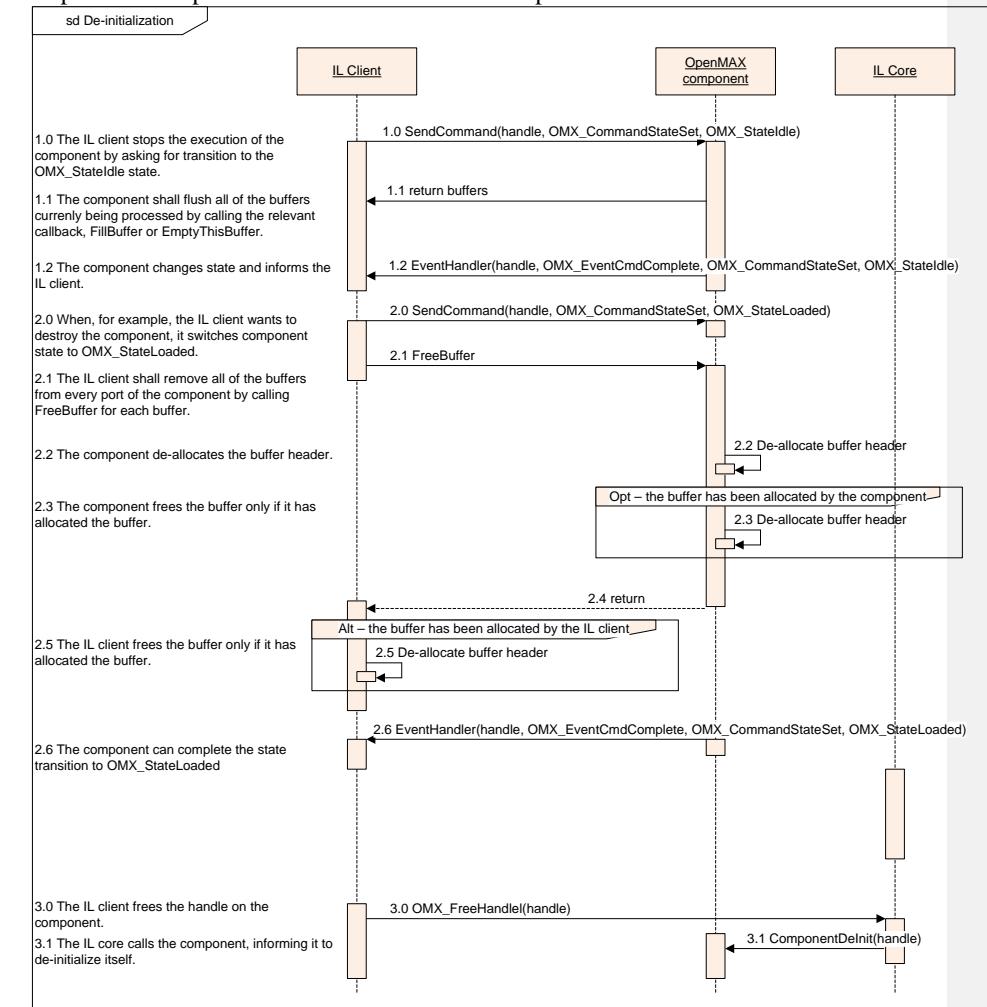
When the IL client decides to stop the execution and dispose of the components, it should first switch the components to the `OMX_StateIdle` state so that all buffers are returned to their suppliers.

When the transition to `OMX_StateIdle` is completed, the IL client can request the component to change its state to `OMX_StateLoaded`. The IL client shall free all of the component's buffers by calling `OMX_FreeBuffer` for each buffer. The `OMX_FreeBuffer` function requires that the component remove the specified buffer from the specified port. If the component allocated the buffer with an `OMX_AllocateBuffer` call, the component shall also free the buffer memory. If the IL client allocated the buffer and assigned it to the component with an

OMX\_UseBuffer call, then the IL client shall de-allocate the buffer memory after calling OMX\_FreeBuffer. This procedure is performed for each non-tunneled port.

When all of the buffers have been freed, the component shall complete the state transition. Finally, the IL client calls the OMX\_FreeHandle function that disposes of the component.

This procedure is performed for each non-tunneled port.



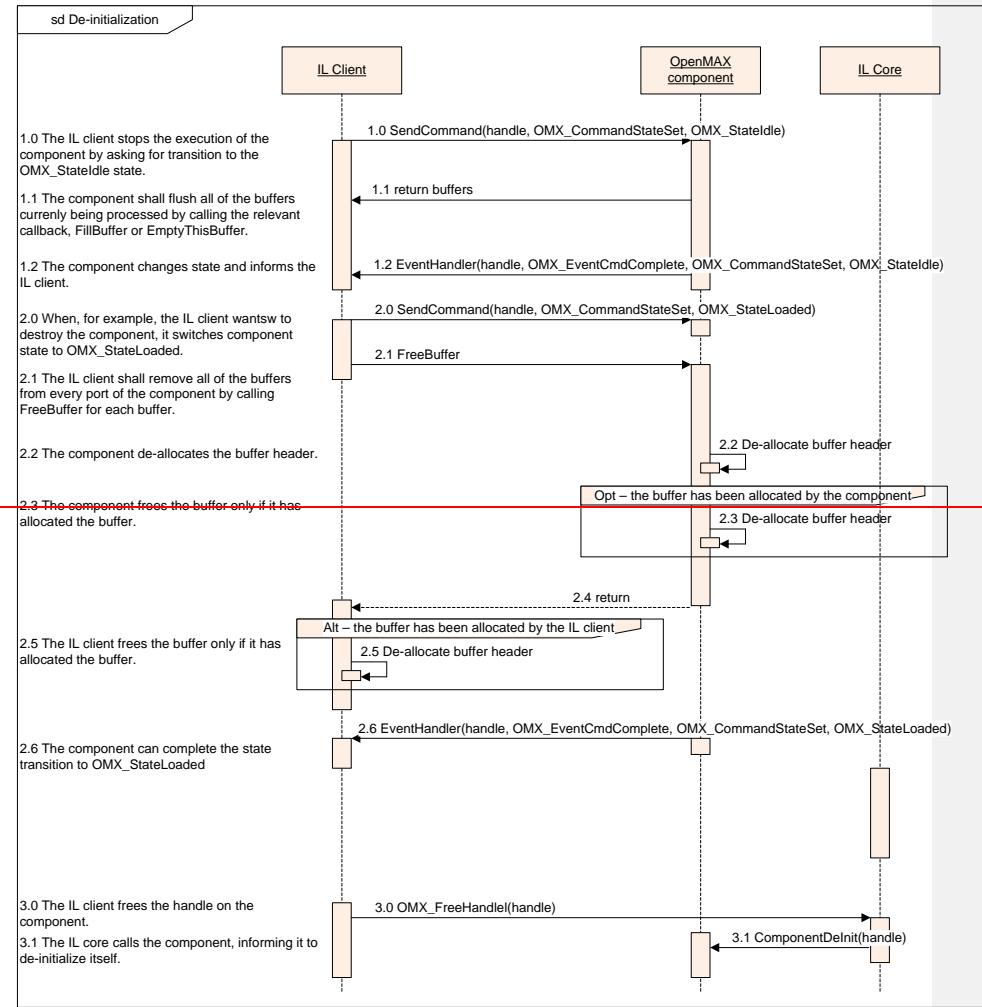
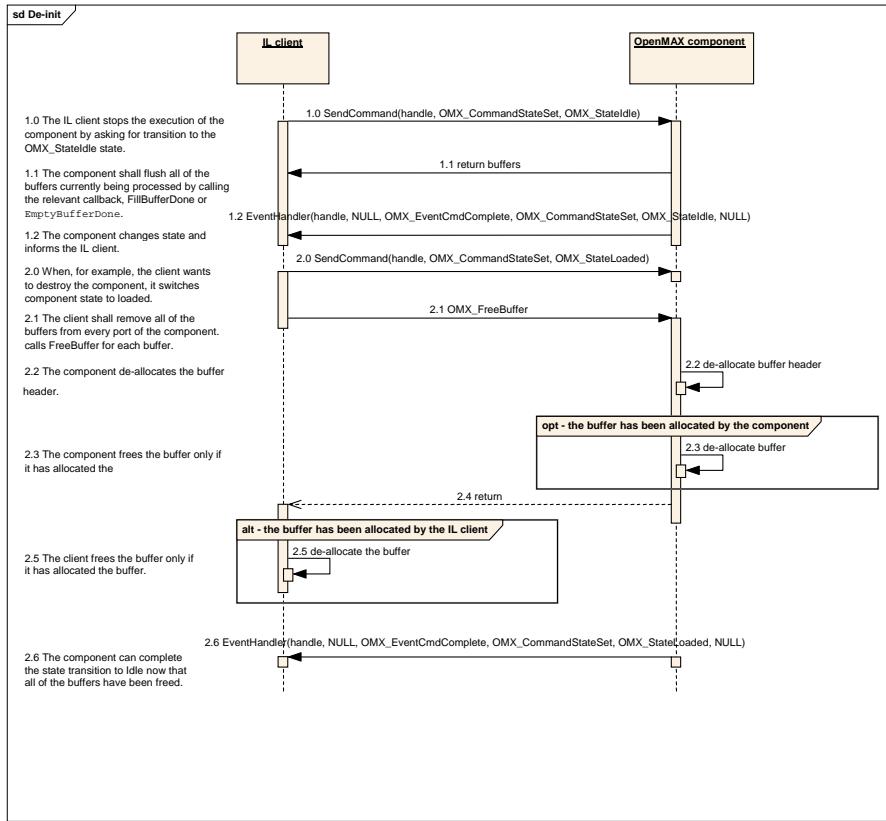
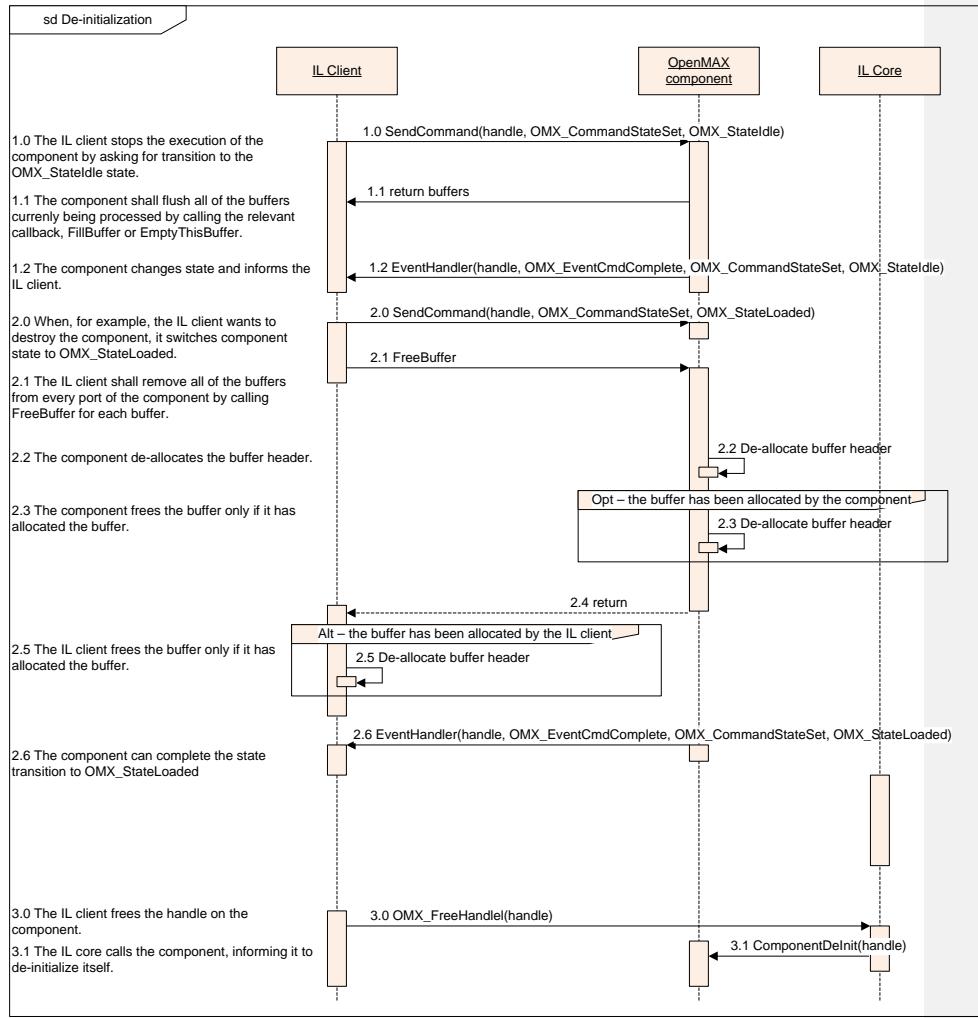


Figure 3-34Figure 3-14

Figure 3-18. illustrates non-tunneled de-initialization





**Figure 3-19. De-initialization of Non-tunneled Components**

A port that is tunneled shall follow the component de-initialization procedure illustrated in Section 3.4.3.2.

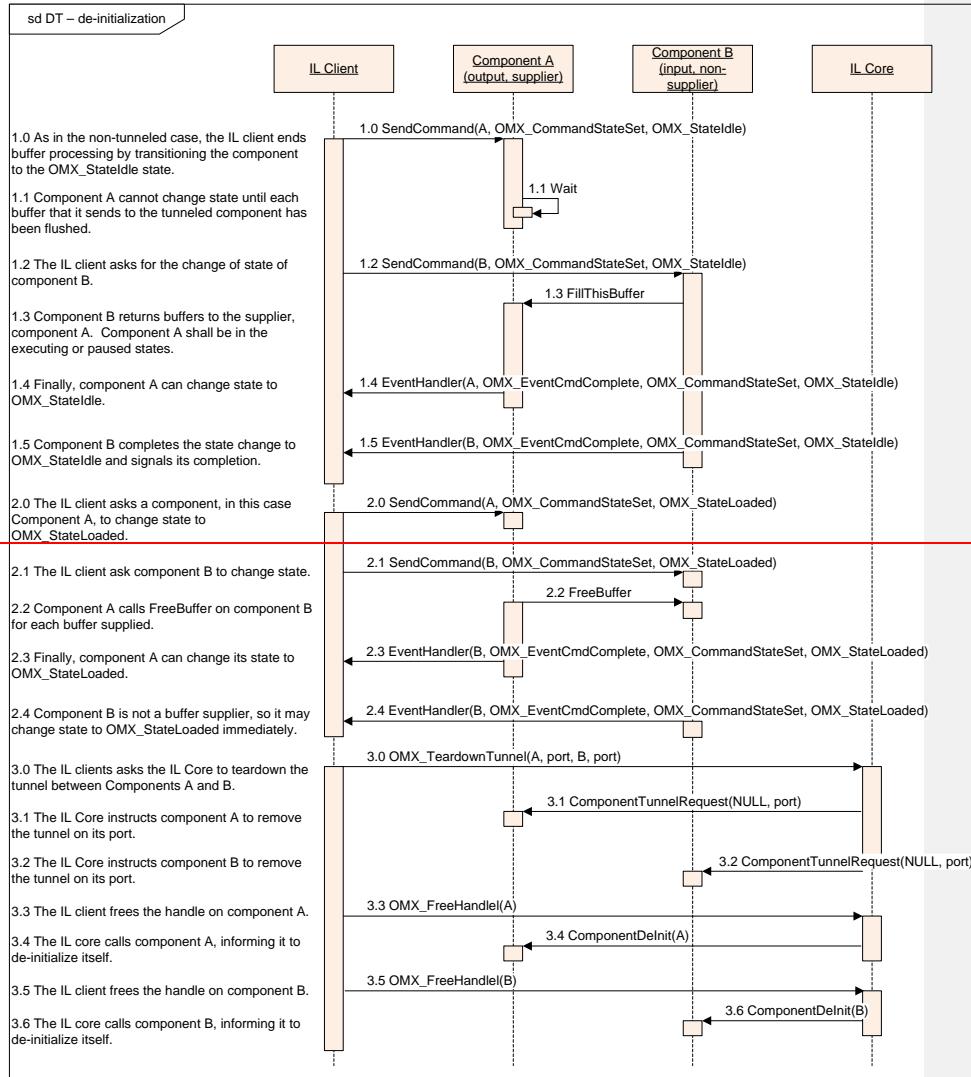
### 3.4.3.2 Tunneled De-Initialization

When the IL client decides to stop the execution and dispose of two components that are tunneled, it may first either disable all ports or transition the component to OMX\_StateLoaded (via OMX\_StateIdle if necessary). The following example illustrates the second method, where it first switches the components to the OMX\_StateIdle state so that all buffers are returned to their suppliers. After all

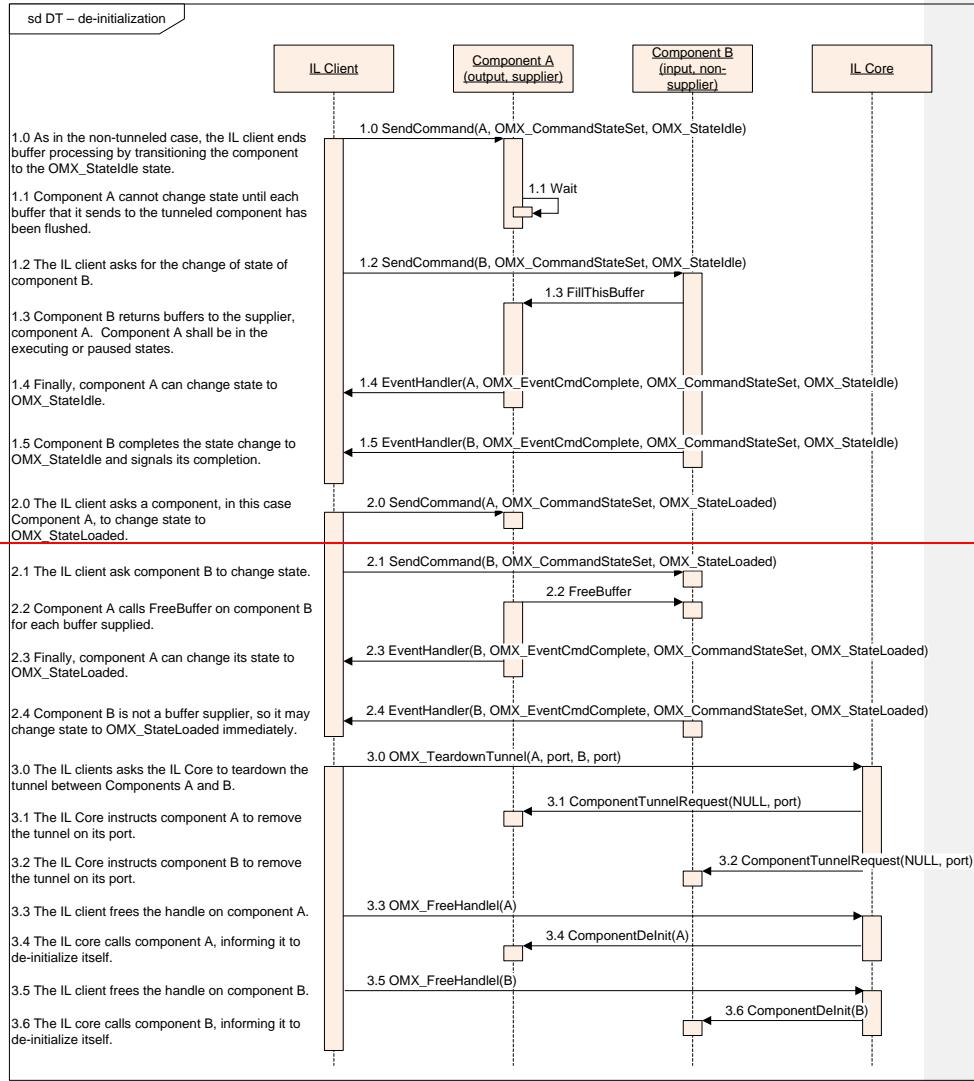
buffers are returned to supplier ports, the components shall complete the transition to OMX\_StateIdle.

The IL client should then start the state transition to OMX\_StateLoaded. When the supplier component frees all the buffers, the components complete the state transition to OMX\_StateLoaded.

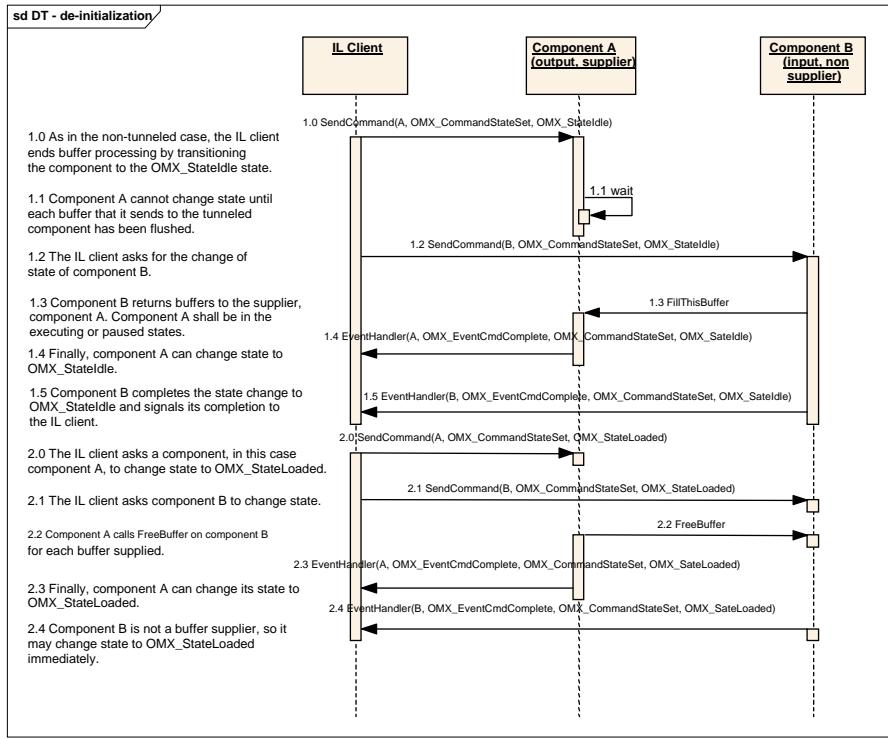
After this point the tunnel shall be disconnected using OMX\_TeardownTunnel, inside this function the IL core will call ComponentTunnelRequest on each component port to disconnect the tunnel. After this has been completed the IL client can call OMX\_FreeHandle to dispose of the components.

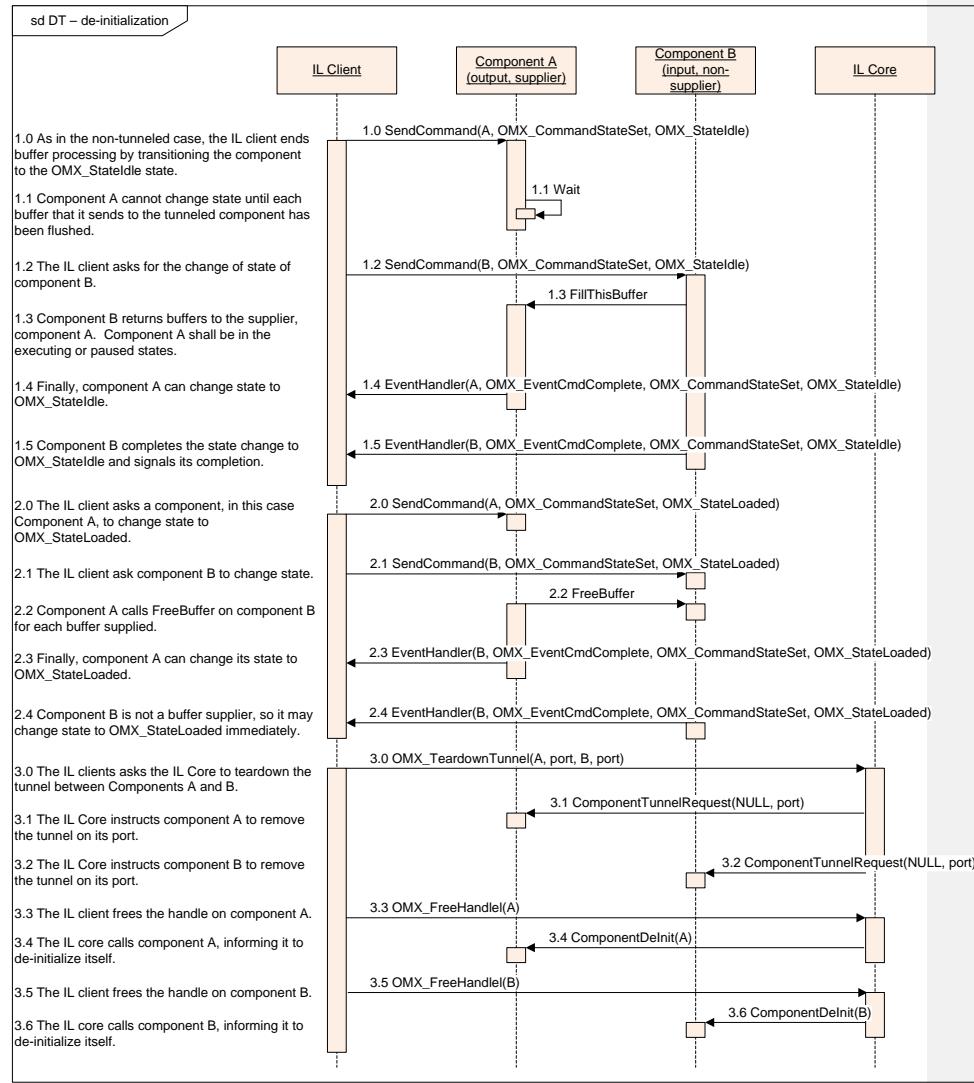


**Figure  
3-22**



**Figure 3-36** **Figure 3-15** **Figure 3-19** illustrates the component de-initialization for a port that is tunneled – the expectation is that the two components have a single port each, and these ports are tunneled together.





**Figure 3-20: De-initialization of Tunneled Components**

### 3.4.4 Port Disablement and Enablement

Disabling a port causes it to behave as if its component transitioned to the OMX\_StateLoaded state. Thus, all of the port's buffers are returned to their suppliers, and any buffers the disabled port allocated are freed. The act of enabling a port inverts this process, putting a port that is effectively in the OMX\_StateLoaded state into the component's state. Thus, if the component is in a state where its ports have buffers, then

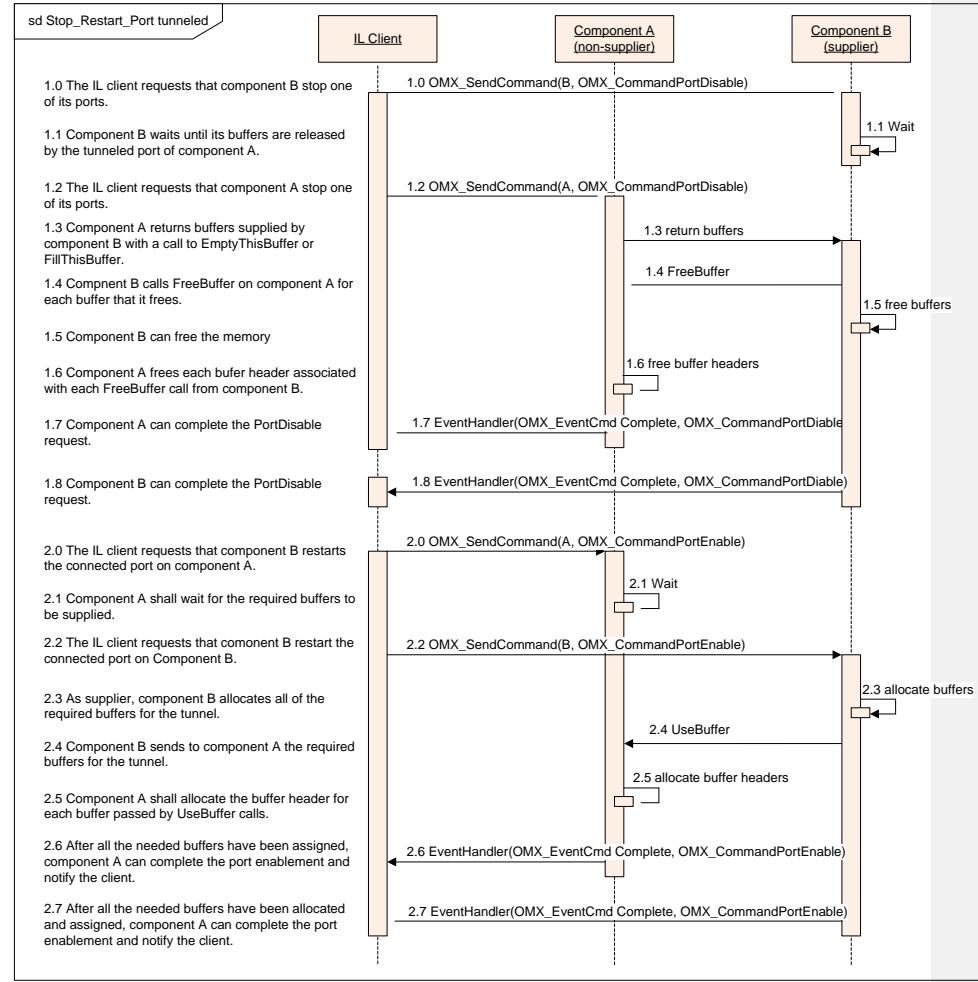
an enabled port will acquire buffers. Likewise, if the component is exchanging buffers, an enabled port will begin exchanging buffers.

Note that if a port is disabled when the component is in the OMX\_StateLoaded state, the port's effective state is still made disjoint from the component's state. Thus, when a component transitions from OMX\_StateLoaded to OMX\_StateIdle, any disabled port will not acquire buffers but, instead, will effectively remain in OMX\_StateLoaded.

The description of port disablement and enablement is divided into tunneling and non-tunneling cases.

#### **3.4.4.1 Tunneled Ports Disablement and Enablement**

[Figure 3-20](#)[Figure 3](#) illustrates the behavior of enabling and disabling tunneled ports.



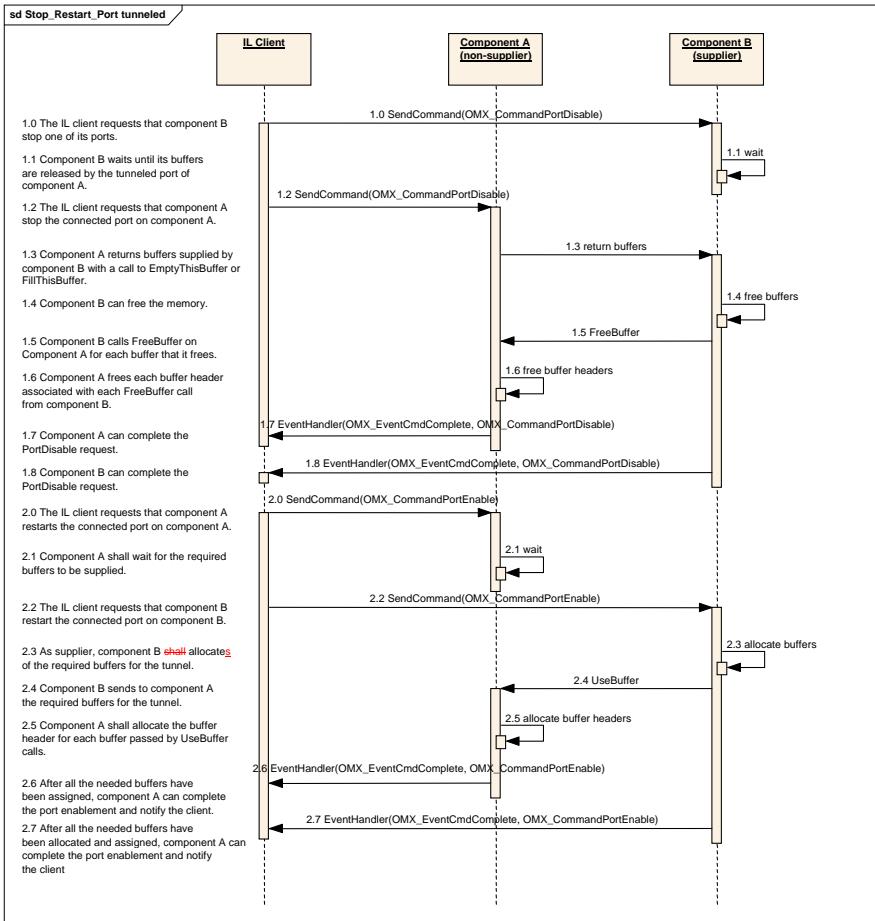
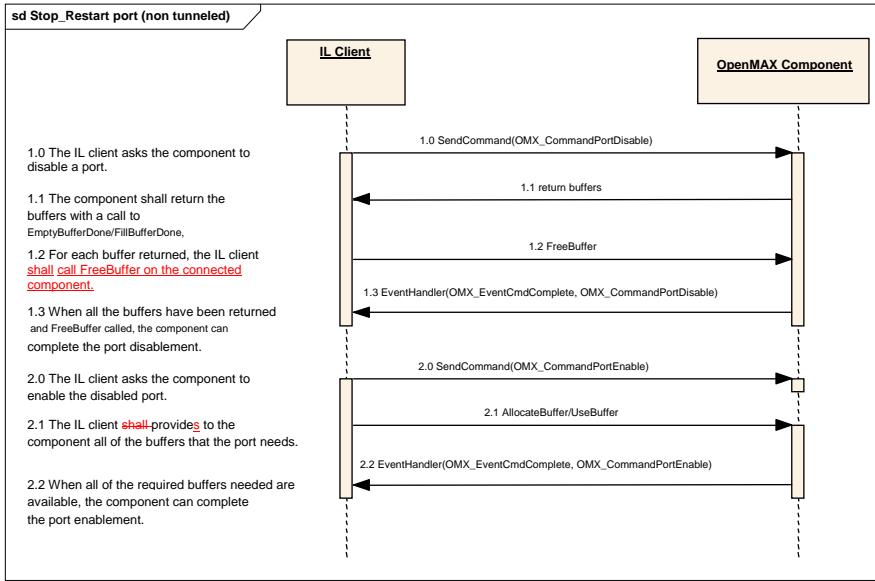


Figure 3-21. Disablement and Enablement of Tunneled Ports

#### 3.4.4.2 Non-tunneled Port Disablement and Enablement

Figure 3-21 illustrates the case of the disablement and enablement procedure for a non-tunneled port. A detailed discussion of `OMX_AllocateBuffer`, `OMX_UseBuffer`, and `OMX_FreeBuffer` is omitted here; for more detailed descriptions of the use of these functions, see sections 3.3.13, 3.3.12, and 3.3.14, respectively.



**Figure 3-22. Disablement and Enablement of Non-tunneled Ports**

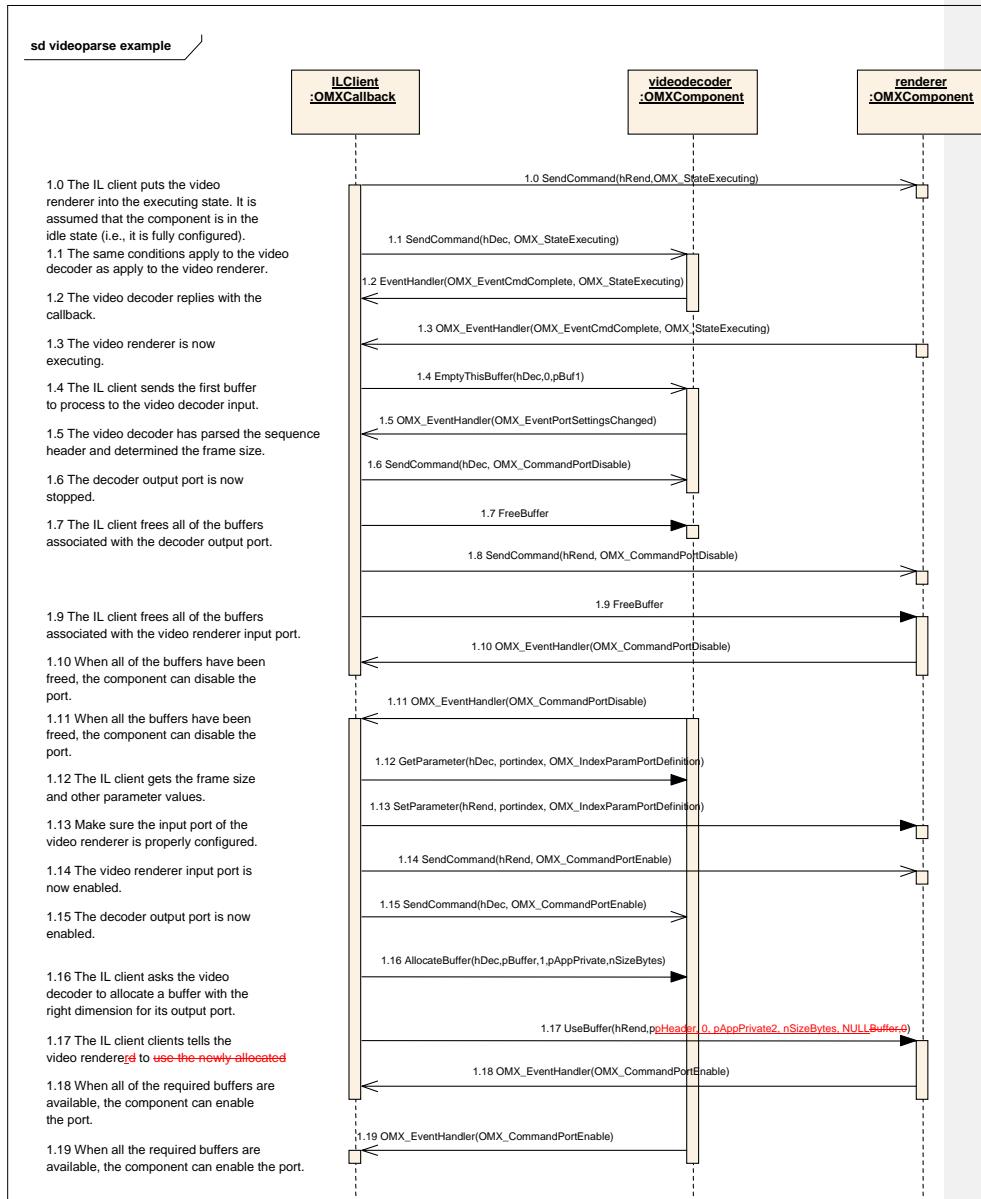
### 3.4.5 Dynamic Port Reconfiguration

This section describes how a component may change its port settings dynamically.

The following examples show where this functionality is typically needed:

- A video decoder parses a sequence header and discovers the frame size of the output pictures, so buffers associated with its output ports shall be rearranged.
- The parameters of an audio stream vary dynamically, and a decoder should change its port settings.

[Figure 3-22](#) shows how a video decoder and a video renderer, both of which exchange data through the IL client, should dynamically change their port settings.



**Figure 3-23. Dynamic Port Reconfiguration**

The sequence starts with the IL client putting a video renderer and a video decoder in the OMX\_StateExecuting state (1.0 through 1.3). At this stage, the output port of the video decoder and the input port of the renderer are not yet configured, since the

dimension of the output frame is unknown *a priori*. The decoder needs to start parsing the input bit stream to derive such information.

In fact, the IL client sends the first buffer to the decoder in step 1.4. Assuming that the video sequence header is included in that first buffer, the OpenMAX IL decoder component will parse it and change its output port settings accordingly.

The OpenMAX IL decoder component shall then notify the IL client by generating the OMX\_EventPortSettingsChanged event (step 1.5). As soon as the IL client receives this callback, it shall disable the output port of the video decoder and the input port of the video renderer (steps 1.6 through 1.11).

The IL client shall then reads the new port settings with OMX\_GetParameterConfig and allocates one or more buffers with the right dimensions for the output port. Once the buffers are allocated, the IL client uses they will be also communicated to the video renderer using OMX\_UseBuffer (1.17). The buffer pointers cannot be pre-announced in this example, because buffers allocated by OMX\_AllocateBuffer (1.16) may change during execution. The input port of the video renderer shall also be set up with OMX\_SetParameterConfig (1.18).

Finally, ports can be enabled and normal processing resumes.

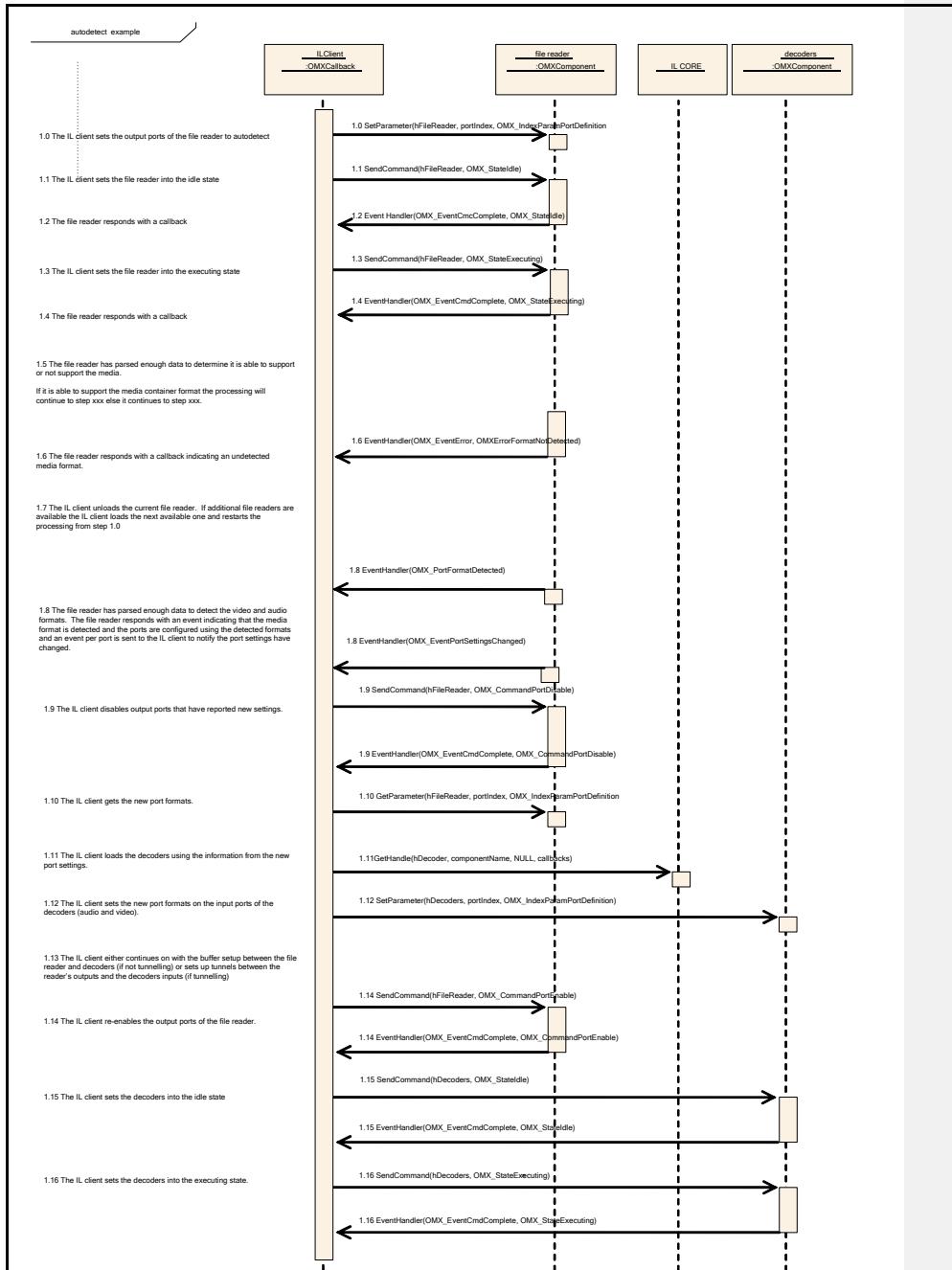
### 3.4.6 Autodetect Port Reconfiguration

This section describes how a component may change its autodetect port settings.

The following examples show where this functionality is typically needed:

- A file reader parses a media container such as a 3GPP file and discovers the video and audio decoders required to decode the elementary streams.
- The encoding types of a media container may vary so a file reader should change its port settings once the formats are determined.

[Figure 3-23 Autodetect Port Reconfiguration](#) shows how a file reader, an audio decoder and a video decoder should connect after the autodetect ports have determined the required port settings.



**Figure 3-24 Autodetect Port Reconfiguration**

The sequence starts with the IL client setting the output port formats (`OMX_IndexParamVideoPortFormat` and `OMX_IndexParamAudioPortFormat`) of the file reader to autodetect.

| Initially ~~only~~ the IL client instantiates only the file reader, lets all output ports communicate with the IL client, and sets all output ports to autodetect. The IL client then commands the file reader to transition into `OMX_StateIdle` thereby allocating all of its buffers. The IL client then commands the file reader to transition into `OMX_StateExecuting`.

The file reader now reads and parses data until it determines if it is able to detect the format of the media container. If the file reader is not able to detect the media container format it will notify the IL client by generating an `OMX_ErrorFormatNotDetected` error (step 1.6). Since the media container format was not detected, the IL client can return to step 1.0 with another file reader component and execute the same sequence. This continues until either the media container format is detected or no more file reader components exist that have not attempted to detect the media container format.

| If the file reader is able to detect the media container format and the ~~the~~ format of the streams it will emit on the output ports, the file reader component will change its output port settings accordingly and notify the IL client by generating the `OMX_EventPortFormatDetected` and `OMX_EventPortSettingsChanged` events (step 1.8) for each output port where the format has been changed. As soon as the IL client receives this callback, it shall disable the changed output ports of the file reader (step 1.9).

| The IL client shall then read the new file reader port settings for all output ports whose settings have changed with `OMX_GetParameterConfig`. Based on ~~the these~~ settings the IL client shall select appropriate decoder components and call the `OMX_GetHandle` function for each. If previous step is successful, valid handles are returned in step 1.11 and the decoder components will be in the `OMX_StateLoaded` state.

The IL client shall configure the decoder components and its ports (including the format settings discovered from the parser). For this purpose, the IL core macro `OMX_SetParameter` shall be used; it may be called multiple times (step 1.12) if needed.

| At this point the IL client may setup the components for either non-tunneled communication (by setting up the buffers itself) or tunneled communication (by setting up tunnels and ~~and~~ letting the components set up the buffers)

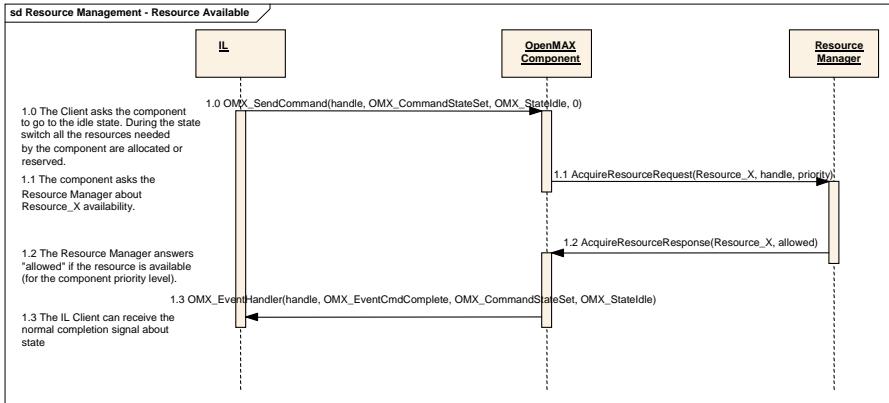
| Finally the IL client re-enables the reader's output ports and transitions the decoders into `OMX_StateIdle` then `OMX_StateExecuting`. At this point processing resumes.

### 3.4.7 Resource Management

This section describes the entry points for resource management. The interface between components and the resource manager are presented only as an example. Only the

interface between the IL client and the components is part of the OpenMAX IL standard definition. An IL client may use the resource manager entry points.

[Figure 3-24](#) proposes the behavior of an IL client is agnostic of the resource manager. The resource manager handles the component internally only, and the IL client has to take no special action.



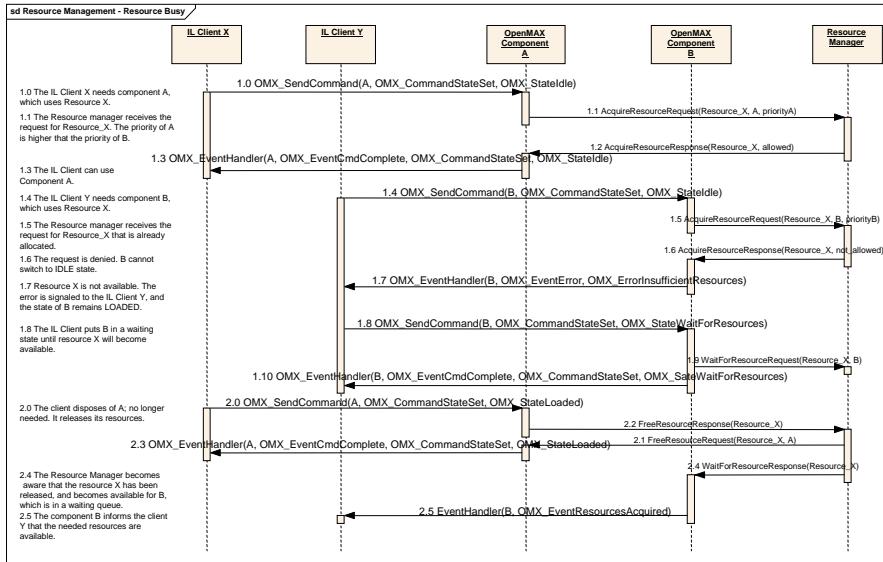
**Figure 3-25. Transition from Loaded to Idle with Resource Management**

In [Figure 3-24](#), the IL client is unaware of the existence of a resource manager. In the implementation of the OpenMAX IL component, an asynchronous call to the resource manager is implemented.

The OpenMAX IL component provides a callback to the resource manager, which receives the signal for the completion of the request.

[Figure 3-24](#) represents a possible implementation of a resource manager, and shows how it can be transparent to the client. The functions `AcquireResourceRequest` and `AcquireResourceResponse` are examples. This specification is concerned only about the interface between the IL client and the components. Details of the interactions between the components and the vendor-specific manager(s) are outside the scope of this specification.

[Figure 3-25](#) presents a more complex use case.



**Figure 3-26. Busy Resource Management**

In [Figure 3-25](#), two different OpenMAX IL components, A and B, need the same resource to work, and they have different priorities. Here, as in the preceding example, the IL clients use the standard transition from OMX\_StateLoaded to OMX\_StateIdle to set up the component and allocate all of the required resources.

The first component, component A, takes ownership of the resource, requesting it from the resource manager. Component A switches to OMX\_StateIdle and is ready to execute.

The second component, component B, asks for the same resource, but in this case the resource manager denies it since a higher priority component, component A, has that resource. This event is reported to the IL client with an error message including the value OMX\_ErrorInsufficientResources. If IL client Y decides that it needs to be notified when this resource becomes available again, it may direct component B to change state to OMX\_StateWaitForResources. This action puts component B in a waiting queue until the resource X will become available. Alternatively, IL client Y may request component B to switch back to OMX\_StateLoaded.

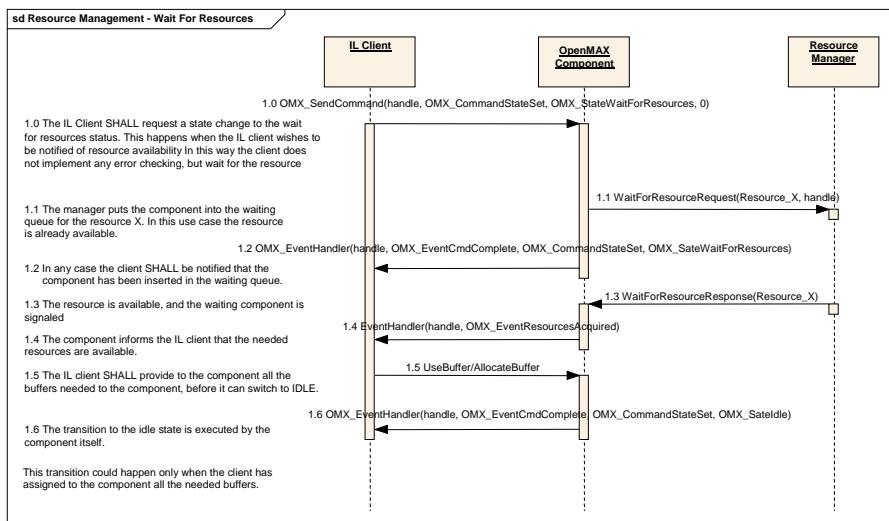
[Figure 3-25](#) also shows the behavior of components when resource X becomes available. Component A changes state to OMX\_StateLoaded and releases all of the resources. The resource manager becomes aware of the available resource and calls Component B, which is already in the waiting queue.

When the resource manager provides the component with all the resources it is waiting on, the component informs the IL client that all resources needed are available with an OMX\_EventResourcesAcquired event. The IL client shall now provide all of the needed buffers to the component. Then, the component can change state by itself to

`OMX_StateIdle` and alert the client about the state change. This waiting queue represents a unique case of automatic state change.

In [Figure 3-25](#), the priorities of components A and B are not compared within the IL layer, and no preemption mechanism is implemented or proposed; an external policy manager, which should communicate with the resource manager, should have this responsibility. The description of such a policy manager is outside the scope of this document and the OpenMAX IL standard in general.

[Figure 3-26](#) presents an example of a client that actively uses the resource management API.



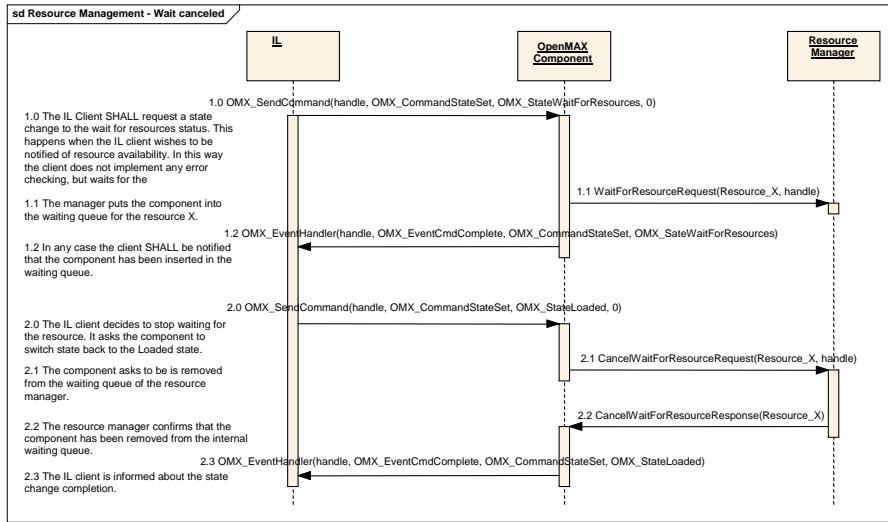
**Figure 3-27. State Change from `OMX_StateLoaded` to `OMX_StateWaitForResources`**

The IL client may request a state change from `OMX_StateLoaded` to `OMX_StateWaitForResources` in case the IL client wants to be notified when the resource becomes available again. For an explanation of `OMX_StateWaitForResources`, see section 3.1.1.2.5.

In this case, the client puts the component into a waiting queue, handled by the resource manager; the change to the idle state happens effectively when the resource will become available or if it is available immediately. In any case, the client receives two different `EventHandler` callbacks that correspond to two different state changes.

The two functions `WaitForResourceRequest` and `WaitForResourceResponse` in [Figure 3-26](#) are not defined in this specification but are examples of an interaction between components and the resource manager.

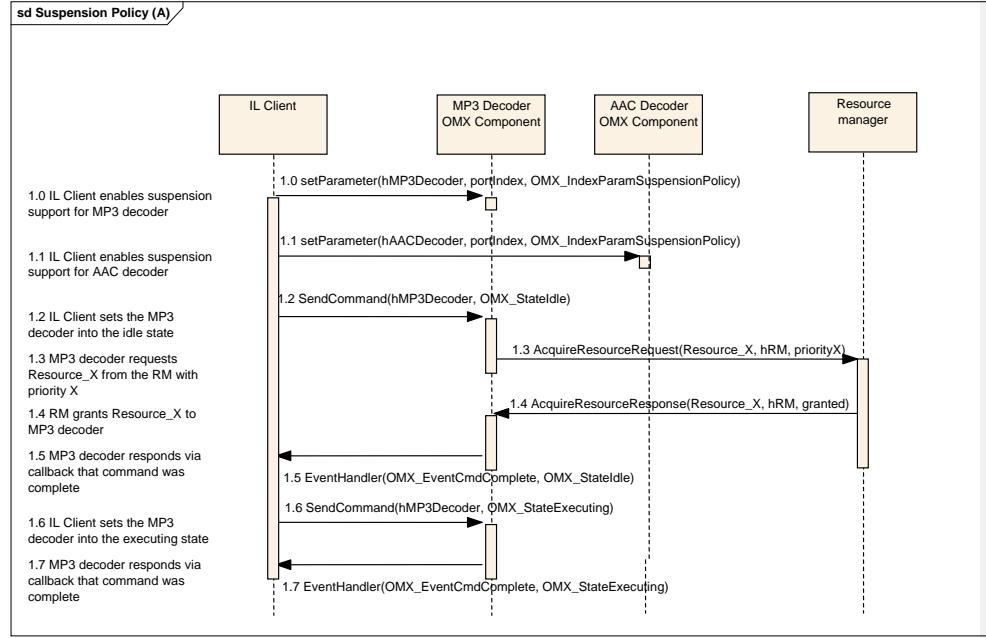
The IL client may decide to stop waiting at a certain time. In this case, it shall request the component to change state back to `OMX_StateLoaded`, as shown in [Figure 3-27](#).



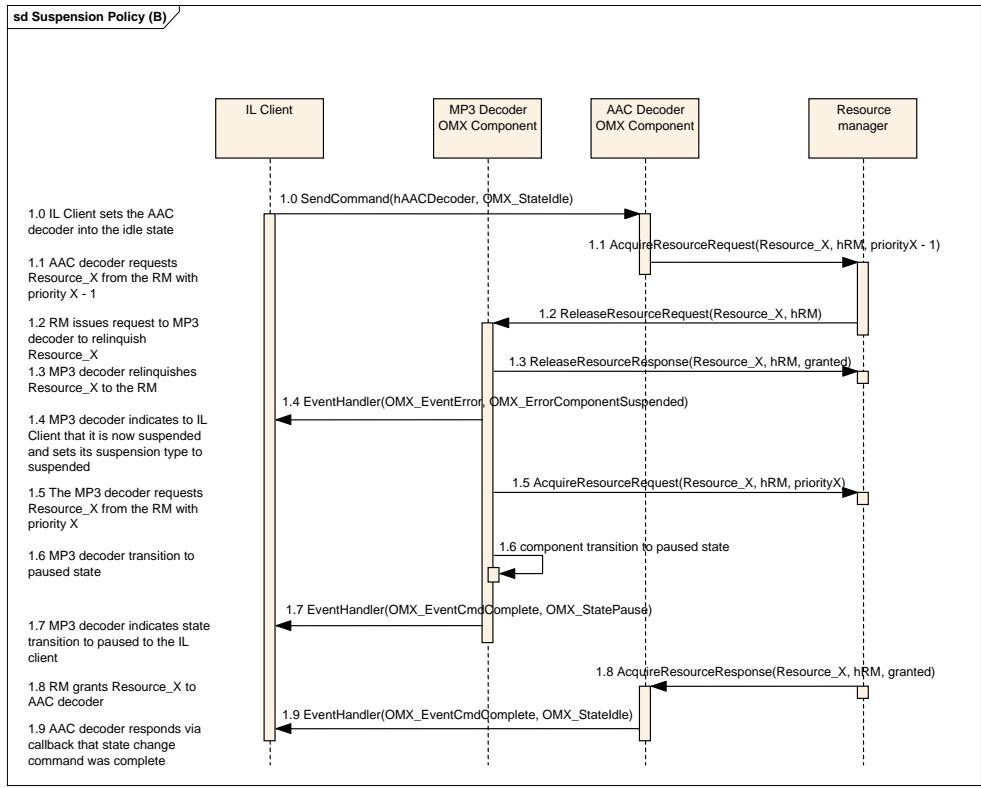
**Figure 3-28. Remove Component from Waiting Status**

### **3.4.8 Component Suspension**

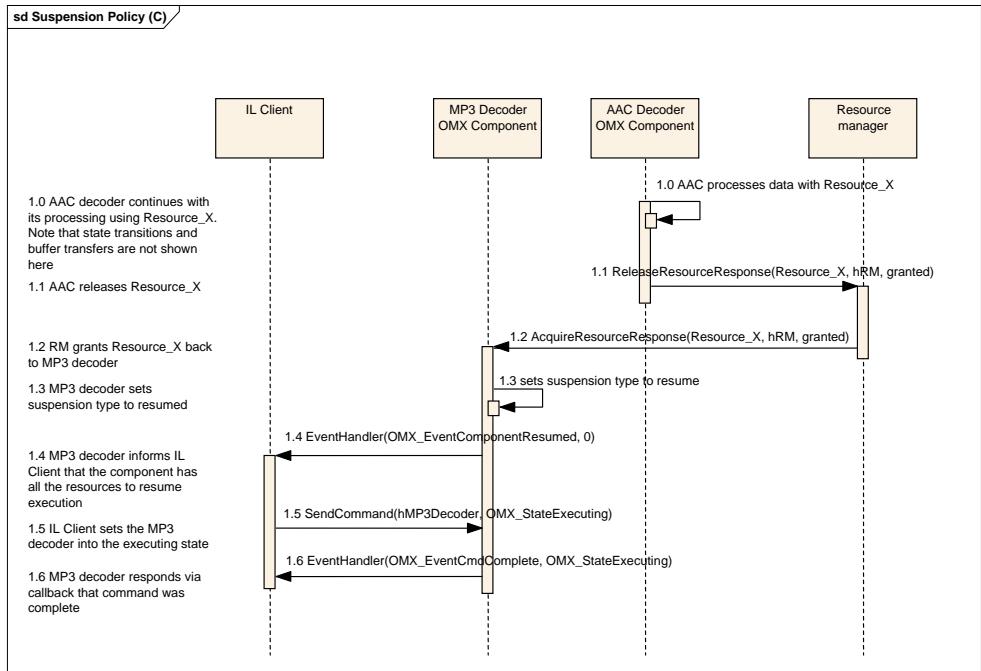
This section shows an example of component suspension due to lack of dynamic resources, and comprise an example of two components, MP3 decoder and AAC decoder, requiring access to a common resource. –



**Figure 3-29: Suspension Policy (A)**



**Figure 3-30: Suspension Policy (B)**



**Figure 3-31: Component Suspension Due to Pre-emption of Resources**

and comprise an example of two components, MP3 decoder and AAC decoder, requiring access to a common resource. The example assume that each component needs to process a set of compressed buffers to be decoded. The IL client sets the components to support the suspension mechanism (1.0 A and 1.1 A) so that any loss of resources while processing the streams can be resumed.

The IL client transitions the MP3 decoder into OMX\_StateIdle (1.2 A). At this time the MP3 decoder issues a request to the resource manager (RM) for Resource\_X (1.3 A). The RM responds to the request by granting Resource\_X to the MP3 decoder (1.4 A). The MP3 decoder is then transitioned to start processing of stream buffers. (Note the buffer transfers are not shown in the diagram for simplicity).

Next the IL client transitions the AAC decoder into OMX\_StateIdle (1.0 B). The AAC decoder issues a request for Resource\_X with as a higher priority client to the RM (1.1 B). The RM in turn issues a request to the MP3 decoder to release Resource\_X (1.2 B). The MP3 decoder complies and releases Resource\_X to the RM (1.3 B).

The MP3 decoder at this point sends an error to the IL client to indicate that the component is suspended (1.4 B). The MP3 decoder issues an acquire resource request for Resource\_X (1.5 B) which of course the RM cannot fulfill since it is a lower priority request but the RM will track this resource request for the MP3 decoder.

The next step for the MP3 decoder is to transition to OMX\_StatePause (1.6 B) and then emit a command complete paused event to the IL client (1.7 B). At this point the MP3 decoder is in a paused suspension state.

Concurrently, the RM may also grant Resource\_X to the AAC decoder after being released by the MP3 decoder (1.7 B). The AAC decoder completes the state change to OMX\_StateIdle by issuing a command complete to the IL client. Assuming the IL client transitions the AAC decoder to executing and after processing a number of buffers (1.0 C) the AAC decoder releases Resource\_X (1.1 C).

The RM then grants Resource\_X to the MP3 component (1.2 C) base on its earlier request (1.5 B). The MP3 decoder then sets its suspension type to resume (1.3 C) and then issues an OMX\_EventComponentResumed message to the IL client (1.4 C). The IL client transitions the MP3 component out of OMX\_StatePause to OMX\_StateExecuting to resume the stream processing (1.5 C-1.6 C).

### **3.5 Slaving Behavior for Port Settings**

Some components have some port settings which are common between their input port and their output port. When these components are not able to perform conversion for these common settings, there is an implicit slaving behavior that the component shall implement to make sure the common settings are always kept the same between both ports.

This slaving behavior is defined as follows:

- when issuing OMX\_SetParameter() on the input port, if the value of the settings in common with the output port are changed, then the component shall update the output port settings and emit a OMX\_EventPortSettingsChanged event on the output port.
- when issuing OMX\_SetParameter() on the output port, the component shall return the error OMX\_ErrorBadParameter if the settings in common with the input port are changed.
- if the settings in common between both ports change during runtime (for example after parsing the stream), then the component shall emit OMX\_EventPortSettingsChanged events on both ports.

Example of components which shall implement this slaving behavior:

- audio decoder, encoder, and processor class have the sampling rate and number of channels in common between input and output ports.
- video decoder, encoder, and IV processor class have the width and height between input and output ports.

## 4 OpenMAX IL Data API

This section describes the typical component usage for the audio, video, image, and other domains. This section also details all of the structures, parameters, and configurations that apply to ports for each of the domains and provides use case examples where appropriate.

### 4.1 Audio

This section describes the structures, parameters, and configuration details for ports in the audio domain. These parameter and configurations details are specified in the OMX\_Audio.h header.

#### 4.1.1 Audio Use Case Examples

Figure 4-1 illustrates an example of an audio playback processing chain. Two sound sources are played simultaneously and are mixed with effects added to both the individual processing paths and the mixed signal. Only OpenMAX IL standard components are shown in this example.

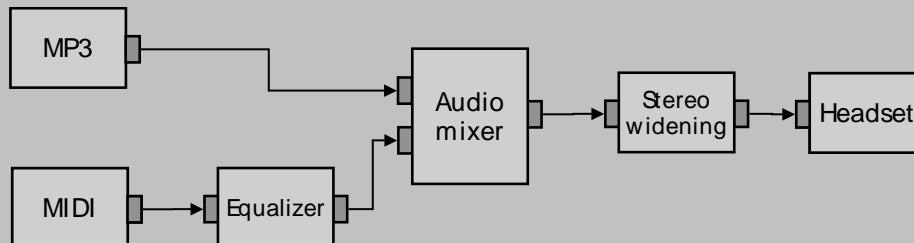


Figure 4-1. Audio Playback Processing Chain

Figure 4-2 illustrates a simple example of speech processing chains with echo cancellation added for an uplink speech path. Speech codecs can be any specified OpenMAX IL codecs.

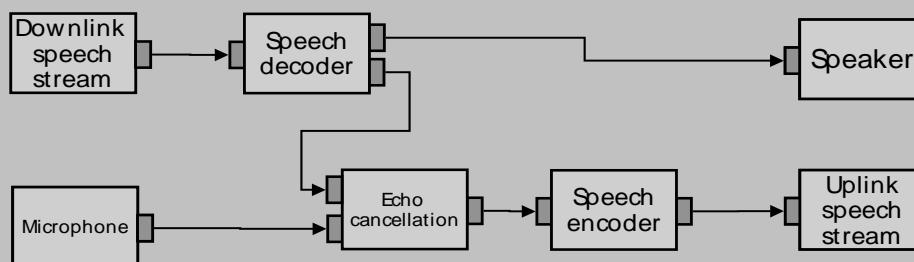


Figure 4-2. Speech Processing Chain

## **4.1.2 Special Issues**

Some audio formats have special or unique requirements that are different from other audio formats, or even from other domains. These issues are described in the following sections.

### **4.1.34.1.2 Minimum Buffer Payload Size for Uncompressed Data**

OpenMAX IL has specified a minimum buffer payload sizes for all types of uncompressed data. The minimum payload size for pulse code modulation (PCM) audio is five milliseconds. This means that an output port of a PCM component shall produce at least five milliseconds of audio data for each buffer. The minimum payload size is applied only for PCM (i.e., OMX\_AUDIO\_CodingPCM) and not for any other formats.

### **4.1.44.1.3 Whole-file Buffering for MIDI Formats**

Most MIDI content formats contain multiple parallel tracks of media data that appear in the file in serial track order rather than interleaved in real-time execution order. In addition, the MIDI state is deterministic only from the beginning of file playback, and thus seeks within any MIDI file require that at least some part of the file be re-processed from the beginning. For these reasons, callers shall provide the full length of the MIDI file data to the [MIDI OpenMAX IL components](#) processing MIDI using the nFileSize field of the OMX\_AUDIO\_PARAM\_MIDITYPE structure. For more information on the OMX\_AUDIO\_PARAM\_MIDITYPE structure, see section [4.1.32](#).

### **4.1.54.1.4 General Enumerations**

OMX\_AUDIO\_CODINGTYPE is the enumeration used to define the possible audio [encoding types](#). If OMX\_AUDIO\_CodingUnused is selected, the coding selection shall be done in a vendor-specific way. Table 4-1 shows the contents of OMX\_AUDIO\_CODINGTYPE.

**Table 4-1: Audio Coding Types**

Field Name	Description	References to Standard(s)
OMX_AUDIO_CodingUnused	Placeholder value when coding is not available	Not available
OMX_AUDIO_CodingAutoDetect	Auto detection of audio format	Not available
OMX_AUDIO_CodingPCM	Any variant of PCM coding	PCM
OMX_AUDIO_CodingADPCM	Any variant of ADPCM encoded data	ADPCM

<b>Field Name</b>	<b>Description</b>	<b>References to Standard(s)</b>
OMX_AUDIO_CodingAMR	Any variant of AMR encoded data	AMR-NB , AMR-WB <u>AMR-WB+</u>
OMX_AUDIO_CodingGSMFR	Any variant of GSM Full-Rate (i.e., GSM610)	GSM-FR
OMX_AUDIO_CodingGSMEFR	Any variant of GSM Enhanced Full-Rate encoded data	GSM-EFR
OMX_AUDIO_CodingGSMHR	Any variant of GSM Half-Rate encoded data	GSM-HR
OMX_AUDIO_CodingPDCFR	Any variant of PDC Full-Rate encoded data	PDC-FR
OMX_AUDIO_CodingPDCEFR	Any variant of PDC Enhanced Full-Rate encoded data	PDC-EFR
OMX_AUDIO_CodingPDCHR	Any variant of PDC Half-Rate encoded data	PDC-HR
OMX_AUDIO_CodingTDMAFR	Any variant of TDMA Full-Rate encoded data (TIA/EIA-136-420)	TDMA-FR
OMX_AUDIO_CodingTDMAEFR	Any variant of TDMA Enhanced Full-Rate encoded data (TIA/EIA-136-410)	TDMA-EFR
OMX_AUDIO_CodingQCELP8	Any variant of QCELP 8 kbps encoded data	QCELP8
OMX_AUDIO_CodingQCELP13	Any variant of QCELP 13 kbps encoded data	QCELP13
OMX_AUDIO_CodingEVRC	Any variant of EVRC encoded data	EVRC
OMX_AUDIO_CodingSMV	Any variant of SMV encoded data	SMV

Field Name	Description	References to Standard(s)
OMX_AUDIO_CodingG711	Any variant of G.711 encoded data	G.711
OMX_AUDIO_CodingG723	Any variant of G.723.1 encoded data	G.723.1
OMX_AUDIO_CodingG726	Any variant of G.726 encoded data	G.726
OMX_AUDIO_CodingG729	Any variant of G.729 encoded data	G.729
OMX_AUDIO_CodingAAC	Any variant of AAC encoded data	MPEG-2 AAC , MPEG-4 AAC HE-AAC v1 , HE-AAC v2
OMX_AUDIO_CodingMP3	Any variant of MP3 encoded data	MPEG-1 Audio , MPEG-2 Audio
OMX_AUDIO_CodingSBC	Any variant of SBC encoded data	SBC
OMX_AUDIO_CodingVORBIS	Any variant of VORBIS encoded data	VORBIS
OMX_AUDIO_CodingWMA	Any variant of WMA encoded data	WMA
OMX_AUDIO_CodingRA	Any variant of RA encoded data	RA
OMX_AUDIO_CodingMIDI	Any variant of MIDI encoded data	<a href="#">SP-MIDISP-MIDISP-MIDI</a> , <a href="#">DLS 1</a> <a href="#">DLS 1</a> <a href="#">DLS +</a> , <a href="#">DLS 2</a> <a href="#">DLS 2</a> <a href="#">DLS 2</a> <a href="#">General MIDI</a> <a href="#">General MIDI</a> , <a href="#">General MIDI 2</a> <a href="#">General MIDI 2</a> , <a href="#">MIDI 2</a> <a href="#">General MIDI 2</a> , <a href="#">GM Lite</a> <a href="#">GM Lite</a> <a href="#">GM Lite</a> , <a href="#">XMF type 0 and 1</a> <a href="#">XMF type 0 and 1</a> <a href="#">XMF type 0 and 1</a> <a href="#">XMF type 0 and 1</a> <a href="#">XMF type 0 and 1</a> <a href="#">XMF type 0 and 1</a> , Mobile XMF

Formatted: Table Text

#### **4.1.64.1.5 Parameter and Configuration Indexes**

The header OMX\_Index.h contains the enumeration OMX\_INDEXTYPE, which contains all standard index values used with the core functions OMX\_GetParameter, OMX\_SetParameter, OMX\_GetConfig, and OMX\_SetConfig. Table 4-2 shows the [index values](#)[indices and associated structures](#) that relate to audio. [The structures are described in the following sections.](#)

**Table 4-2: Audio Coding Types by Index-specific indices and associated structures.**

OpenMAX IL Indices ( OMX_Index.h )	Corresponding OpenMAX IL Audio Structures ( OMX_Audio.h )
<a href="#">OMX_IndexParamPortDefinition</a>	<a href="#">OMX_PARAM_PORTDEFINITIONTYPE</a> with <a href="#">OMX_AUDIO_PORTDEFINITIONTYPE</a>
OMX_IndexParamAudioPortFormat	OMX_AUDIO_PARAM_PORTFORMATTYPE
OMX_IndexParamAudioPcm	OMX_AUDIO_PARAM_PCMMODETYPE
OMX_IndexParamAudioMp3	OMX_AUDIO_PARAM_MP3TYPE
OMX_IndexParamAudioAac	OMX_AUDIO_PARAM_AACPROFILETYPE
OMX_IndexParamAudioVorbis	OMX_AUDIO_PARAM_VORBISTYPE
OMX_IndexParamAudioWma	OMX_AUDIO_PARAM_WMATYPE
OMX_IndexParamAudioRa	OMX_AUDIO_PARAM_RATYPE
OMX_IndexParamAudioSbc	OMX_AUDIO_PARAM_SBCTYPE
OMX_IndexParamAudioAdpcm	OMX_AUDIO_PARAM_ADPCMTYPE
OMX_IndexParamAudioG723	OMX_AUDIO_PARAM_G723TYPE
OMX_IndexParamAudioG726	OMX_AUDIO_PARAM_G726TYPE
OMX_IndexParamAudioG729	OMX_AUDIO_PARAM_G729TYPE
OMX_IndexParamAudioAmr	OMX_AUDIO_PARAM_AMRTYPE
OMX_IndexParamAudioGsm_FR	OMX_AUDIO_PARAM_GSMFRTYPE
OMX_IndexParamAudioGsm_EFR	OMX_AUDIO_PARAM_GSMEFRTYPE
OMX_IndexParamAudioGsm_HR	OMX_AUDIO_PARAM_GSMHRTYPE
OMX_IndexParamAudioTdma_FR	OMX_AUDIO_PARAM_TDMAFRTYPE
OMX_IndexParamAudioTdma_EFR	OMX_AUDIO_PARAM_TDMAEFRTYPE
OMX_IndexParamAudioPdc_FR	OMX_AUDIO_PARAM_PDCFRTYPE
OMX_IndexParamAudioPdc_EFR	OMX_AUDIO_PARAM_PDCEFRTYPE
OMX_IndexParamAudioPdc_HR	OMX_AUDIO_PARAM_PDCHRTYPE
OMX_IndexParamAudioQcelp8	OMX_AUDIO_PARAM_QCELP8TYPE
OMX_IndexParamAudioQcelp13	OMX_AUDIO_PARAM_QCELP13TYPE
OMX_IndexParamAudioEvrc	OMX_AUDIO_PARAM_EVRCTYPE
OMX_IndexParamAudioSmv	OMX_AUDIO_PARAM_SMVTYPE
OMX_IndexParamAudioMidi	OMX_AUDIO_PARAM_MIDITYPE

<b>OpenMAX IL Indices ( OMX_Index.h )</b>	<b>Corresponding OpenMAX IL Audio Structures ( OMX_Audio.h )</b>
OMX_IndexParamAudioMidiLoadUserSound	OMX_AUDIO_PARAM_MIDILOADUSERSOUNDTYPE
OMX_IndexConfigAudioMidiImmediateEvent	OMX_AUDIO_CONFIG_MIDIIMMEDIATEEVENTTYPE
OMX_IndexConfigAudioMidiSoundBankProgram	OMX_AUDIO_CONFIG_MIDISOUNDBANKPROGRAMTYPE
OMX_IndexConfigAudioMidiControl1	OMX_AUDIO_CONFIG_MIDICONTROLTYPE
OMX_IndexConfigAudioMidiStatus	OMX_AUDIO_CONFIG_MIDISTATUSTYPE
OMX_IndexConfigAudioMidiMetaEvent	OMX_AUDIO_CONFIG_MIDIMETAEVENTTYPE
OMX_IndexConfigAudioMidiMetaEventData	OMX_AUDIO_CONFIG_MIDIMETAEVENTDATATYPE
OMX_IndexConfigAudioVolume	OMX_AUDIO_CONFIG_VOLUMETYPE
OMX_IndexConfigAudioChannelVolume	OMX_AUDIO_CONFIG_CHANNELVOLUMETYPE
OMX_IndexConfigAudioBalance	OMX_AUDIO_CONFIG_BALANCETYPE
OMX_IndexConfigAudioMute	OMX_AUDIO_CONFIG_MUTETYPE
OMX_IndexConfigAudioChannelMute	OMX_AUDIO_CONFIG_CHANNELMUTETYPE
OMX_IndexConfigAudioLoudness	OMX_AUDIO_CONFIG_LOUDNESSTYPE
OMX_IndexConfigAudioBass	OMX_AUDIO_CONFIG_BASSTYPE
OMX_IndexConfigAudioTreble	OMX_AUDIO_CONFIG_TREBLETYPE
OMX_IndexConfigAudioEqualizer	OMX_AUDIO_CONFIG_EQUALIZERTYPE
OMX_IndexConfigAudioStereoWidening	OMX_AUDIO_CONFIG_STEREOWIDENINGTYPE
OMX_IndexConfigAudioChorus	OMX_AUDIO_CONFIG_CHORUSTYPE
OMX_IndexConfigAudioReverberation	OMX_AUDIO_CONFIG_REVERBERATIONTYPE
OMX_IndexConfigAudioEchoCancellation	OMX_AUDIO_CONFIG_ECHOCANCELLATIONTYPE
OMX_IndexConfigAudioNoiseReduction	OMX_AUDIO_CONFIG_NOISEREDUCTIONTYPE
<u>OMX_IndexConfigAudio3DOutput</u>	<u>OMX_AUDIO_CONFIG_3DOUTPUTTYPE</u>
<u>OMX_IndexConfigAudio3DLocation</u>	<u>OMX_AUDIO_CONFIG_3DLOCATIONTYPE</u>
<u>OMX_IndexParamAudio3DDopplerMode</u>	<u>OMX_AUDIO_PARAM_3DDOPPLERMODETYPE</u>
<u>OMX_IndexConfigAudio3DDopplersSettings</u>	<u>OMX_AUDIO_CONFIG_3DDOPPLERSETTINGSTYPE</u>
<u>OMX_IndexConfigAudio3DLevels</u>	<u>OMX_AUDIO_CONFIG_3DLEVELSTYPE</u>

OpenMAX IL Indices ( OMX_Index.h )	Corresponding OpenMAX IL Audio Structures ( OMX_Audio.h )
<u><a href="#">OMX_IndexConfigAudio3DDistanceAttenuation</a></u>	<u><a href="#">OMX_AUDIO_CONFIG_3DDISTANCEATTENUATIONTYPE</a></u>
<u><a href="#">OMX_IndexConfigAudio3DDirectivitySettings</a></u>	<u><a href="#">OMX_AUDIO_CONFIG_3DDIRECTIVITYSETTINGSTYPE</a></u>
<u><a href="#">OMX_IndexConfigAudio3DDirectivityOrientation</a></u>	<u><a href="#">OMX_AUDIO_CONFIG_3DDIRECTIVITYORIENTATIONTYPE</a></u>
<u><a href="#">OMX_IndexConfigAudio3DMacrosopicOrientation</a></u>	<u><a href="#">OMX_AUDIO_CONFIG_3DMACROSCOPICORIENTATIONTYPE</a></u>
<u><a href="#">OMX_IndexConfigAudio3DMacrosopicSize</a></u>	<u><a href="#">OMX_AUDIO_CONFIG_3DMACROSCOPICSIZETYPE</a></u>
<u><a href="#">OMX_IndexParamAudioQueryChannelMapping</a></u>	<u><a href="#">OMX_AUDIO_CHANNELMAPPINGTYPE</a></u>
<u><a href="#">OMX_IndexConfigAudioSbcBitpool</a></u>	<u><a href="#">OMX_AUDIO_SBCBITPOOLTYPE</a></u>
<u><a href="#">OMX_IndexConfigAudioAmrMode</a></u>	<u><a href="#">OMX_AUDIO_AMRMODETYPE</a></u>
<u><a href="#">OMX_IndexConfigAudioBitrate</a></u>	<u><a href="#">OMX_AUDIO_CONFIG_BITRATETYPE</a></u>
<u><a href="#">OMX_IndexConfigAudioAMRISFIIndex</a></u>	<u><a href="#">OMX_AUDIO_CONFIG_AMRISFTYPE</a></u>
<u><a href="#">OMX_IndexParamAudioFixedPoint</a></u>	<u><a href="#">OMX_AUDIO_FIXEDPOINTTYPE</a></u>

#### [4.1.74.1.6 OMX\\_AUDIO\\_PORTDEFINITIONTYPE](#)

The OMX\_AUDIO\_PORTDEFINITIONTYPE structure is used to define all of the parameters necessary for the compliant component to set up an input or an output audio path. If additional information is needed to define the parameters of the port, such as frequency, additional structures such as the OMX\_AUDIO\_PARAM\_PCMMODETYPE structure shall be sent to supply the extra parameters for the port. The number of audio paths for input and output will vary by the type of the audio component.

OMX\_Component.h contains common port definition structures for all media domains.

The OMX\_AUDIO\_PORTDEFINITIONTYPE structure can query the current ~~or default~~ definition of an audio port or set the definition of an audio port for a component. The OMX\_AUDIO\_PORTDEFINITIONTYPE structure is included as part of the OMX\_PARAM\_PORTDEFINITIONTYPE structure, it is accessed via the OMX\_GetParameter function or the OMX\_SetParameter function using the OMX\_IndexParamPortDefinition index.

OMX\_AUDIO\_PORTDEFINITIONTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PORTDEFINITIONTYPE {
    OMX_STRING eMIMETYPE;
    OMX_NATIVE_DEVICETYPE pNativeRender;
    OMX_BOOL bFlagErrorConcealment;
    OMX_AUDIO_CODINGTYPE eEncoding;
```

```
} OMX_AUDIO_PORTDEFINITIONTYPE;
```

The parameters for OMX\_AUDIO\_PORTDEFINITIONTYPE are defined as follows.

- ~~cMIMETYPE is the MIME type of data for the port. If a MIME type string buffer is not supplied this parameter shall be set to NULL.~~
- pNativeRender is the platform-specific reference for an output device; otherwise this field is 0.
- bFlagErrorConcealment turns on error concealment if it is supported by the OpenMAX IL component.
- eEncoding is the type of data expected for this port (e.g., PCM, AMR, MP3, and so forth).

#### **4.1.84.1.7 OMX\_AUDIO\_PARAM\_PORTFORMATTYPE**

OMX\_AUDIO\_PARAM\_PORTFORMATTYPE is the structure for the port format parameter. This structure enumerates the various data ~~input/output~~ formats that the port supports.

This parameter call can be used with both OMX\_GetParameter and OMX\_SetParameter. In the OMX\_GetParameter case, the caller specifies all fields and the OMX\_GetParameter call returns the value of eEncoding. The value of nIndex goes from 0 to N-1, where N is the number of formats supported by the port. The port does not need to report N as the caller can determine N by enumerating all the formats supported by the port. Each port shall support at least one format. If there are no more formats, OMX\_GetParameter returns OMX\_ErrorNoMore (i.e., nIndex is supplied where the value is N or greater). Ports supply formats in order of preference: Higher preference formats are provided with lower values of nIndex.

For OMX\_SetParameter, the nIndex field is ~~nIndex~~ ignored. If the format is supported, it is set as the format of the port, and the default values for the format are programmed into the port definition type as a side effect. This allows the caller to query the default values for the format without having to know them in advance.

OMX\_AUDIO\_PARAM\_PORTFORMATTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_PORTFORMATTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nIndex;
    OMX_AUDIO_CODINGTYPE eEncoding;
} OMX_AUDIO_PARAM_PORTFORMATTYPE;
```

The parameters for OMX\_AUDIO\_PARAM\_PORTFORMATTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the port represents the port that this structure applies to.~~
- nIndex indicates the enumeration index for the format from ~~0x0~~ to N-1.

- eEncoding is the type of data expected for this port (e.g., PCM, AMR, MP3, and so forth).

#### **4.1.94.1.8 OMX\_AUDIO\_PARAM\_PCMMODETYPE**

The OMX\_AUDIO\_PARAM\_PCMMODETYPE structure is used to set or query the current **or default** settings for PCM audio using the OMX\_SetParameter function. It is also used to set the parameters for PCM audio using the OMX\_SetParameter function. When calling either the OMX\_SetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioPcm.

Note that the minimum buffer payload size is applied to all modes of PCM audio. The payload size is defined by OMX\_MIN\_PCMPAYLOAD\_MSEC and is five milliseconds.

OMX\_AUDIO\_PARAM\_PCMMODETYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_PCMMODETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_NUMERICALDATATYPE eNumData;
    OMX_ENDIANTYPE eEndian;
    OMX_BOOL bInterleaved;
    OMX_U32 nBitPerSample;
    OMX_U32 nSamplingRate;
    OMX_AUDIO_PCMMODETYPE ePCMMMode;
    OMX_AUDIO_CHANNELTYPE eChannelMapping[OMX_AUDIO_MAXCHANNELS];
} OMX_AUDIO_PARAM_PCMMODETYPE;
```

##### **4.1.9.14.1.8.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_PCMMODETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo, multi-channel).
- eNumData indicates whether the PCM data is signed or unsigned.

**Table 4-3: OMX\_NUMERICALDATATYPE enumeration**

<b>Field Name</b>	<b>Description</b>
<a href="#">OMX_NumericalDataSigned</a>	Signed data
<a href="#">OMX_NumericalDataUnsigned</a>	Unsigned data

- eEndian indicates whether PCM data is in little- or big-endian order.

**Table 4-4: OMX\_ENDIANTYPE enumeration**

<b>Field Name</b>	<b>Description</b>
<a href="#">OMX_EndianBig</a>	Big-endian data

Field Name	Description
<a href="#">OMX_EndianLittle</a>	Little-endia data

- `bInterleaved` indicates whether the data is normal interleaved or non-interleaved. True represents normal interleaved data, and false represents non-interleaved data such as block data.
- `nBitPerSample` is the number of bits per sample.
- `nSamplingRate` is the sampling rate of the source data. Use the value 0 for variable or unknown sampling rate.
- `ePCMMode` is the PCM mode enumeration. [Table 4-5](#) identifies the PCM mode.

**Table 4-5: PCM Mode**

Field Name	Description
<a href="#">OMX_AUDIO_PCMModeLinear</a>	Linear PCM encoded data
<a href="#">OMX_AUDIO_PCMModeALaw</a>	A law PCM encoded data (G.711)
<a href="#">OMX_AUDIO_PCMModeMULaw</a>	$\mu$ law PCM encoded data (G.711)

- `eChannelMapping` is the audio channel mapping enumeration. A component will indicate the order of the audio channels as shown in [Table 4-6](#). A component should use the default channel mapping (standard RIFF/WAV mapping as present in standard multi-channel WAV files: FRONT\_LEFT FRONT\_RIGHT FRONT\_CENTER LOW\_FREQUENCY BACK\_LEFT BACK\_RIGHT .) if possible.

**Table 4-6: Audio Channel Mapping**

Field Name	Description
<a href="#">OMX_AUDIO_ChannelNone</a>	Unused or empty
<a href="#">OMX_AUDIO_ChannelUnknown</a>	<a href="#">Unknown channel mapping</a>
<a href="#">OMX_AUDIO_ChannelLF</a>	Left front
<a href="#">OMX_AUDIO_ChannelRF</a>	Right front
<a href="#">OMX_AUDIO_ChannelCF</a>	Center front
<a href="#">OMX_AUDIO_ChannelLS</a>	Left surround
<a href="#">OMX_AUDIO_ChannelRS</a>	Right surround
<a href="#">OMX_AUDIO_ChannelLFE</a>	Low frequency effects
<a href="#">OMX_AUDIO_ChannelCS</a>	Back surround
<a href="#">OMX_AUDIO_ChannelLR</a>	Left rear
<a href="#">OMX_AUDIO_ChannelRR</a>	Right rear
<a href="#">OMX_AUDIO_ChannelLCF</a>	<a href="#">Left of center front</a>
<a href="#">OMX_AUDIO_ChannelRCF</a>	<a href="#">Right of center front</a>
<a href="#">OMX_AUDIO_ChannelLHS</a>	<a href="#">Left (Hand) side</a>
<a href="#">OMX_AUDIO_ChannelRHS</a>	<a href="#">Right (Hand) side</a>

Field Name	Description
<a href="#">OMX_AUDIO_ChannelCCT</a>	<a href="#">Center top</a>
<a href="#">OMX_AUDIO_ChannelFLT</a>	<a href="#">Front left top</a>
<a href="#">OMX_AUDIO_ChannelFCT</a>	<a href="#">Front center top</a>
<a href="#">OMX_AUDIO_ChannelFRT</a>	<a href="#">Front right top</a>
<a href="#">OMX_AUDIO_ChannelBLT</a>	<a href="#">Back left top</a>
<a href="#">OMX_AUDIO_ChannelBCT</a>	<a href="#">Back center top</a>
<a href="#">OMX_AUDIO_ChannelBRT</a>	<a href="#">Back right top</a>

#### 4.1.9.24.1.8.2 Functionality

The OMX\_AUDIO\_PARAM\_PCMODETYPE structure sets the parameters of PCM audio.

#### 4.1.104.1.9 OMX\_AUDIO\_PARAM\_MP3TYPE

The OMX\_AUDIO\_PARAM\_MP3TYPE structure is used to set or query the current ~~or default~~ settings for the MPEG Layer-3 (MP3) codec component using the OMX\_GetParameter function. It is also used to set the parameters of the MP3 codec component using the OMX\_SetParameter function. The index specified for this structure is OMX\_IndexParamAudioMp3 when calling either the OMX\_GetParameter or the OMX\_SetParameter functions.

OMX\_AUDIO\_PARAM\_MP3TYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_MP3TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_U32 nBitRate;
    OMX_U32 nSampleRate;
    OMX_U32 nAudioBandWidth;
    OMX_AUDIO_CHANNELMODETYPE eChannelMode;
    OMX_AUDIO_MP3STREAMFORMATTYPE eFormat;
} OMX_AUDIO_PARAM_MP3TYPE;
```

#### 4.1.10.14.1.9.1 Parameter Definitions

The parameters for OMX\_AUDIO\_PARAM\_MP3TYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo, multi-channel).
- nBitRate is the bit rate of the encoded MP3 audio. If the bit rate is variable or unknown, this parameter has the value 0.
- nSampleRate is the sample rate of the encoded or decoded audio.

- nAudioBandWidth is the audio bandwidth in Hz to which an encoder should limit the audio signal. Use the value 0 to let encoder decide.
- eChannelMode is the enumeration of OMX\_AUDIO\_CHANNELMODETYPE for the audio channel mode. AAC and MP3, and SBC use this value, although the names are more appropriate for MP3. [Table 4-7](#) shows the values.

**Table 4-7: Audio Channel Mode**

Mode	Description
OMX_AUDIO_ChannelModeStereo	Two channels. The bit rate allocation between the two channels changes according to each channel's information.
OMX_AUDIO_ChannelModeJointStereo	A mode that takes advantage of what is common between the two channels for higher compression gain.
OMX_AUDIO_ChannelModeDual	Two mono channels. Each channel is encoded with half the bit rate of the overall bit rate.
OMX_AUDIO_ChannelModeMono	Mono channel mode.

- eFormat is the stream format type supported for encoding and decoding MP3 content. [Table 4-8](#) shows the possible MP3 audio stream format types for OMX\_AUDIO\_MP3STREAMFORMATTYPE.

**Table 4-8: MP3 Stream Format Values**

Field Name	Description
<a href="#">OMX_AUDIO_MP3StreamFormatMP1Layer3</a>	<a href="#">MPEG1 Layer 3 stream format.</a>
<a href="#">OMX_AUDIO_MP3StreamFormatUnknown</a>	<a href="#">Unknown, unused or not required format setting.</a>
<a href="#">OMX_AUDIO_MP3StreamFormatMP1Layer3</a>	<a href="#">MPEG1 Layer 3 stream format.</a>
OMX_AUDIO_MP3StreamFormatMP2Layer3	MPEG2 Layer 3 stream format.
OMX_AUDIO_MP3StreamFormatMP2_5Layer3	MPEG2.5 Layer 3 stream format.

#### 4.1.10.24.1.9.2 Functionality

The OMX\_AUDIO\_PARAM\_MP3TYPE structure sets the parameters of the MP3 codec.

#### **4.1.114.1.10 OMX\_AUDIO\_PARAM\_AACPROFILETYPE**

The OMX\_AUDIO\_PARAM\_AACPROFILETYPE structure is used to set or query the current ~~or default~~ settings for the MPEG AAC codec component using the OMX\_GetParameter function. It is also used to set the parameters of the AAC codec component using the OMX\_SetParameter function. The index specified for this structure is OMX\_IndexParamAudioAac when calling either the OMX\_GetParameter or the OMX\_SetParameter functions.

OMX\_AUDIO\_PARAM\_AACPROFILETYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_AACPROFILETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_U32 nSampleRate;
    OMX_U32 nBitRate;
    OMX_U32 nAudioBandWidth;
    OMX_U32 nFrameLength;
    OMX_U32 nAACtools;
    OMX_U32 nAACERTools;
    OMX_AUDIO_AACPROFILETYPE eAACProfile;
    OMX_AUDIO_AACSTREAMFORMATTYPE eAACStreamFormat;
    OMX_AUDIO_CHANNELMODETYPE eChannelMode;
} OMX_AUDIO_PARAM_AACPROFILETYPE;
```

#### **4.1.114.1.10.1 Parameter Definitions**

The parameters for the OMX\_AUDIO\_PARAM\_AACPROFILETYPE structure are defined as follows.

- nPortIndex ~~represents the port that this structure applies to is a read-only value containing the index of the port.~~
- nChannels is the number of channels of audio (mono, stereo, multi-channel).
- nSampleRate is the sample rate of the encoded or decoded audio.
- nBitRate is the bit rate of the encoded AAC audio. If the bit rate is variable or unknown, this parameter has the value 0.
- nAudioBandWidth is the audio bandwidth in Hz to which an encoder should limit the audio signal. Use the value 0 to let the encoder decide.
- nFrameLength is the frame length of the codec in audio samples per channel. The value can be 1024 (AAC) or 960 (AAC-LC), 2048 (HE-AAC), 512 or 480 (AAC-LD). Use the value 0 to let encoder decide.
- nAACtools is the AAC tool usage. [Table 4-9](#) shows the preprocessor defines that should be used to signal the use of AAC coding tools. Use OMX\_AUDIO\_AACToolAll to let the encoder decide. Preprocessor defines are

used to allow parameter passing in the following fashion:

```
"AACtoolParam = OMX_AUDIO_AACToolMS + OMX_AUDIO_AACToolTNS;"
```

**Table 4-9: AAC Tool Usage**

Define Name	Description
OMX_AUDIO_AACToolNone	No AAC tools allowed (encoder configuration) or active (optional decoder information output).
OMX_AUDIO_AACToolMS	Mid/Side (MS) joint coding tool.
OMX_AUDIO_AACToolIS	Intensity Stereo (IS) tool.
OMX_AUDIO_AACToolTNS	Temporal Noise Shaping (TNS) tool.
OMX_AUDIO_AACToolPNS	MPEG-4 Perceptual Noise Substitution (PNS) tool.
OMX_AUDIO_AACToolLTP	MPEG-4 Long Term Prediction (LTP) tool.
OMX_AUDIO_AACToolAll	All AAC tools allowed or active.

- nAACERtools is the AAC Error Resilience tool usage. [Table 4-10](#) shows the preprocessor defines that should be used to signal the use of AAC Error Resilience tools. Use OMX\_AUDIO\_AACERAll to let encoder decide. Preprocessor defines are used to allow parameter passing in the following fashion:

```
"AACERtoolParam = OMX_AUDIO_AACERRVLC + OMX_AUDIO_AACERHCR;"
```

**Table 4-10: AAC Error Resilience Tool Usage**

Define Name	Description
OMX_AUDIO_AACERNone	No AAC ER tools allowed/used
OMX_AUDIO_AACERVCB11	Virtual Code Books for AAC section data (VCB11)
OMX_AUDIO_AACERRVLC	Reversible Variable Length Coding (RVLC)
OMX_AUDIO_AACERHCR	Huffman Codeword Reordering (HCR)
OMX_AUDIO_AACERAll	All AAC ER tools allowed/used

- eAACProfile is the enumeration of OMX\_AUDIO\_AACPFILETYPE for the AAC profile type. The term *profile* is used in the MPEG-2 AAC standard and the terms *object type* and *profile* are used in the MPEG-4 AAC standard. [Table 4-11](#) shows the values and descriptions.

**Table 4-11: AAC Profile Type**

Field Name	Description
<a href="#">OMX_AUDIO_AACObjectNull</a>	<a href="#">Null - not used</a>
<a href="#">OMX_AUDIO_AACObjectUnknown</a>	<a href="#">Unknown, unused or not required setting.</a>
<a href="#">OMX_AUDIO_AACObjectNull</a>	<a href="#">Null - not used</a>

Field Name	Description
OMX_AUDIO_AACObjectMain	AAC Main object/profile
OMX_AUDIO_AACObjectLC	AAC Low Complexity object/profile (MPEG-4: AAC profile)
OMX_AUDIO_AACObjectSSR	AAC Scalable Sample Rate object/profile
OMX_AUDIO_AACObjectLTP	AAC Long Term Prediction object
OMX_AUDIO_AACObjectHE	High Efficiency AAC (object type SBR, MPEG-4: HE-AAC profile)
OMX_AUDIO_AACObjectScalable	AAC Scalable object
OMX_AUDIO_AACObjectERLC	ER AAC Low Complexity object (Error Resilient AAC-LC)
OMX_AUDIO_AACObjectLD	AAC Low Delay object (Error Resilient)
OMX_AUDIO_AACObjectHE_PS	AAC High Efficiency with Parametric Stereo coding (HE-AAC v2, object type PS)

- eAACStreamFormat is the enumeration of OMX\_AUDIO\_AACSTREAMFORMATTYPE for the AAC stream format. [Table 4-12](#) shows the field names and values.

**Table 4-12: AAC Stream Format Type**

Field Name	Description
OMX_AUDIO_AACStreamFormatMP2ADTS	MPEG-2 AAC Audio Data Transport Stream format
OMX_AUDIO_AACStreamFormatUnknown	Unknown, unused or not required format setting.
OMX_AUDIO_AACStreamFormatMP2ADTS	MPEG-2 AAC Audio Data Transport Stream format
OMX_AUDIO_AACStreamFormatMP4ADTS	MPEG-4 AAC Audio Data Transport Stream format
OMX_AUDIO_AACStreamFormatMP4LOAS	Low Overhead Audio Stream format
OMX_AUDIO_AACStreamFormatMP4LATM	Low Overhead Audio Transport Multiplex
OMX_AUDIO_AACStreamFormatADIF	Audio Data Interchange Format
OMX_AUDIO_AACStreamFormatMP4FF	AAC inside MPEG-4/ISO File Format
OMX_AUDIO_AACStreamFormatRAW	AAC Raw Format (access units)

- eChannelMode is the enumeration for the audio channel mode used by AAC and MP3, although the names are more appropriate for MP3. For more information on MP3, see section [4.1.9](#).

#### **4.1.11.24.1.10.2     Functionality**

The OMX\_AUDIO\_PARAM\_AACPROFILETYPE structure sets the parameters of the AAC codec.

#### **4.1.124.1.11       OMX\_AUDIO\_PARAM\_VORBISTYPE**

The OMX\_AUDIO\_PARAM\_VORBISTYPE structure is used to set or query the current ~~or default~~ settings for the Vorbis codec component of the Ogg Vorbis format using the OMX\_GetParameter function. It is also used to set the parameters of the Vorbis codec component using the OMX\_SetParameter function. The index specified for this structure is OMX\_IndexParamAudioVorbis when calling either the OMX\_GetParameter or the OMX\_SetParameter functions.

OMX\_AUDIO\_PARAM\_VORBISTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_VORBISTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_U32 nBitRate;
    OMX_U32 nMinBitRate;
    OMX_U32 nMaxBitRate;
    OMX_U32 nSampleRate;
    OMX_U32 nAudioBandWidth;
    OMX_S32 nQuality;
    OMX_BOOL bManaged;
    OMX_BOOL bDownmix;
} OMX_AUDIO_PARAM_VORBISTYPE;
```

#### **4.1.12.14.1.11.1   Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_VORBISTYPE are defined as follows.

- nPortIndex ~~is the read only value containing the index of the portrepresents the port that this structure applies to.~~
- nChannels is the number of channels of audio (mono, stereo, multi-channel).
- nBitRate is the bit rate of the encoded Vorbis audio. If the bit rate is variable or unknown, this parameter has the value 0. Encoding is set to the bit rate closest to the specified value in bits per second (bps).
- nMinBitRate sets the minimum bit rate in bps.
- nMaxBitRate sets the maximum bit rate in bps.

- `nSampleRate` is the sample rate of the encoded or decoded audio. Use the value 0 for variable or unknown sampling rate.
- `nAudioBandWidth` is the audio bandwidth in Hz to which an encoder should limit the audio signal. Use the value 0 to let encoder decide.
- `nQuality` sets the encoding quality between -1 (low) and 10 (high). In the default mode of operation, the quality level is 3. The normal quality range is 0-10.
- `bManaged` sets the bit rate management mode. This turns off the normal variable bit rate (VBR) encoding but allows the encoder to enforce hard or soft bit rate constraints. This mode can be slower and may also be of lower quality; it is primarily useful for streaming.
- `bDownmix` sets the downmix input from stereo to mono. This parameter has no effect on non-stereo streams. This parameter is useful for lower bit-rate encoding.

#### **4.1.12.24.1.11.2     Functionality**

The `OMX_AUDIO_PARAM_VORBISTYPE` structure sets the parameters of the Vorbis codec.

#### **4.1.134.1.12       OMX\_AUDIO\_PARAM\_WMATYPE**

The `OMX_AUDIO_PARAM_WMATYPE` structure is used to set or query the current ~~or default~~ settings for the Windows Media® audio codec component using the `OMX_GetParameter` function. It is also used to set the parameters of the Windows Media audio codec component using the `OMX_SetParameter` function. When calling either the `OMX_GetParameter` or the `OMX_SetParameter` functions, the index specified for this structure is `OMX_IndexParamAudioWma`.

`OMX_AUDIO_PARAM_WMATYPE` is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_WMATYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U16 nChannels;
    OMX_U32 nBitRate;
    OMX_AUDIO_WMAFORMATTYPPE eFormat;
    OMX_AUDIO_WMAPROFILETYPE eProfile;
    OMX_U32 nSamplingRate;
    OMX_U16 nBlockAlign;
    OMX_U16 nEncodeOptions;
    OMX_U32 nSuperBlockAlign;
    OMX_U32 nBitsPerSample;
    OMX_U32 nAdvancedEncodeOpt;
    OMX_U32 nAdvancedEncodeOpt2 ;
} OMX_AUDIO_PARAM_WMATYPE;
```

#### **4.1.13.14.1.12.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_WMATYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo).
- nBitRate is the bit rate of the encoded Windows Media audio. If the bit rate is variable or unknown, this parameter has a value 0.
- eFormat is the enumeration for the version of the Windows Media audio codec. [Table 4-13](#) shows the field names and values.

**Table 4-13: Windows Media Audio Codec Version**

Field Name	Description
<a href="#">OMX_AUDIO_WMAFormatUnused</a>	<a href="#">The version of the Windows Media audio codec is either not applicable or is unknown.</a>
<a href="#">OMX_AUDIO_WMAFormatUnknown</a>	<a href="#">Unknown, unused or not required format setting.</a>
<a href="#">OMX_AUDIO_WMAFormatUnused</a>	<a href="#">The version of the Windows Media audio codec is either not applicable or is unknown.</a>
OMX_AUDIO_WMAFormat7	Windows Media audio version 7.
OMX_AUDIO_WMAFormat8	Windows Media audio version 8.
OMX_AUDIO_WMAFormat9	Windows Media audio version 9.
<a href="#">OMX_AUDIO_WMAFormat9_Professional</a>	<a href="#">Windows Media audio version 9 Professional.</a>
<a href="#">OMX_AUDIO_WMAFormat9_Lossless</a>	<a href="#">Windows Media audio version 9 Lossless.</a>
<a href="#">OMX_AUDIO_WMAFormat9_LosslessOMX_AUDIO_WMAFormatMax</a>	<a href="#">Windows Media audio version 9 Lossless. For future versions of Windows Media audio codecs.</a>
<a href="#">OMX_AUDIO_WMAFormat9_Voice</a>	<a href="#">Windows Media audio version 9 Voice.</a>
<a href="#">OMX_AUDIO_WMAFormat10_Professional</a>	<a href="#">Windows Media audio version 10 Professional.</a>
<a href="#">OMX_AUDIO_WMAFormat10_Voice</a>	<a href="#">Windows Media audio version 10 Voice.</a>

- eProfile is the enumeration for the profile of the Windows Media audio codec. [Table 4-14](#) shows the field names and values.

**Table 4-14: Windows Media Audio Codec Profile**

Field Name	Description
<u>OMX_AUDIO_WMAProfileUnused</u>	<u>The profile of the Windows Media audio codec is either not applicable or is unknown.</u>
<u>OMX_AUDIO_WMAProfileUnknown</u>	<u>Unknown, unused or not required setting.</u>
<u>OMX_AUDIO_WMAProfileUnused</u>	<u>The profile of the Windows Media audio codec is either not applicable or is unknown.</u>
OMX_AUDIO_WMAProfileL1	Windows Media audio version 9 profile L1.
OMX_AUDIO_WMAProfileL2	Windows Media audio version 9 profile L2.
OMX_AUDIO_WMAProfileL3	Windows Media audio version 9 profile L3.
<u>OMX_AUDIO_WMAProfileM0</u>	<u>Windows Media audio Pro profile M0.</u>
<u>OMX_AUDIO_WMAProfileM1</u>	<u>Windows Media audio Pro profile M1.</u>
<u>OMX_AUDIO_WMAProfileM2</u>	<u>Windows Media audio Pro profile M2.</u>
<u>OMX_AUDIO_WMAProfileM3</u>	<u>Windows Media audio Pro profile M3.</u>
<u>OMX_AUDIO_WMAProfileN1</u>	<u>Windows Media audio Lossless profile N1.</u>
<u>OMX_AUDIO_WMAProfileN2</u>	<u>Windows Media audio Lossless profile N2.</u>
<u>OMX_AUDIO_WMAProfileN3</u>	<u>Windows Media audio Lossless profile N3.</u>
<u>OMX_AUDIO_WMAProfileS1</u>	<u>Windows Media audio Voice profile S1.</u>
<u>OMX_AUDIO_WMAProfileS2</u>	<u>Windows Media audio Voice profile S2.</u>

- nSamplingRate is the sampling rate of the source data.
- nBlockAlign is the block alignment, or block size, in bytes of the audio codec.
- nEncodeOptions is WMA Type-specific data.
- nSuperBlockAlign is WMA Type-specific data.
- nBitsPerSample refers to the stream can be encoded for 24-bit or 16-bit. This parameter is required to support wma lossless AND wma pro configuration. This parameter is specified as a per channel value.
- nAdvancedEncodeOpt is WMA Type-specific data. It refers to bit packed words indicating the features supported for LBR bitstream. This parameter is valid for OMX\_AUDIO\_WMAFormat9\_Lossless wma lossless and OMX\_AUDIO\_WMAFormat10\_Professional for both encoders and decoders.
- nAdvancedEncodeOpt2 is WMA Type-specific data. It refers to bit packed words indicating the features supported for LBR bitstream. This parameter is valid for OMX\_AUDIO\_WMAFormat9\_Lossless wma lossless and OMX\_AUDIO\_WMAFormat10\_Professional for both encoders and decoders.

#### **4.1.144.1.13 OMX\_AUDIO\_PARAM\_RATYPE**

The OMX\_AUDIO\_PARAM\_RATYPE structure is used to set or query the current ~~or default~~ settings for the RealAudio® codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioRa.

OMX\_AUDIO\_PARAM\_RATYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_RATYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nChannels;  
    OMX_U32 nSamplingRate;  
    OMX_U32 nBitsPerFrame;  
    OMX_U32 nSamplePerFrame;  
    OMX_U32 nCouplingQuantBits;  
    OMX_U32 nCouplingStartRegion;  
    OMX_U32 nNumRegions;  
    OMX_AUDIO_RAFORMATTYPE eFormat;  
} OMX_AUDIO_PARAM_RATYPE;
```

#### **4.1.14.14.1.13.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_RATYPE are defined as follows.

- ~~nPortIndex: is the read-only value containing the index of the port represents the port that this structure applies to.~~
- nChannels is the number of audio channels.
- nSamplingRate is the sampling rate of the source data.
- nBitsPerFrame is the value for bits per frame.
- nSamplePerFrame is the value for samples per frame.
- nCouplingQuantBits is the number of coupling quantization bits in the stream.
- nCouplingStartRegion is the coupling start region in the stream.
- nNumRegions is the number of regions value.
- eFormat is the audio format. [Table 4-15](#) shows the possible RealAudio format types in OMX\_AUDIO\_RAFORMATTYPE. See <https://community.helixcommunity.org/realcodecs/#RealAudio> for further details.

**Table 4-15: Supported RealAudio Format Types**

Field Name	RA Format Descriptions
<del>OMX_AUDIO_RAFormatUnused</del>	<del>Format unused or unknown</del>

Field Name	RA Format Descriptions
<a href="#">OMX_AUDIO_RAFormatUnknown</a>	<a href="#">Unknown, unused or not required setting.</a>
<a href="#">OMX_AUDIO_RAFormatUnused</a>	<a href="#">Format unused or unknown</a>
OMX_AUDIO_RA8	RealAudio 8 audio codec
OMX_AUDIO_RA9	RealAudio 9 audio codec
OMX_AUDIO_RA10_AAC	MPEG-4 AAC codec for bitrates of more than 128kbps
OMX_AUDIO_RA10_CODEC	RealAudio codec for bitrates less than 128 kbps
OMX_AUDIO_RA10_LOSSLESS	RealAudio Lossless
OMX_AUDIO_RA10_MULTICHANNEL	RealAudio Multichannel
OMX_AUDIO_RA10_VOICE	RealAudio Voice for bitrates below 15 kbps.

#### 4.1.14.24.1.13.2     Functionality

The OMX\_AUDIO\_PARAM\_RATYPE structure sets the parameters of the RealAudio codec.

#### 4.1.154.1.14       OMX\_AUDIO\_PARAM\_SBCTYPE

The Subband codec (SBC) is a mandatory audio codec for applications that support the Bluetooth™ Advance Audio Distribution Profile (A2DP). The A2DP codec algorithm is designed to obtain high quality audio at medium bit rates with a low computational complexity.

The OMX\_AUDIO\_PARAM\_SBCTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioSbc.

OMX\_AUDIO\_PARAM\_SBCTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_SBCTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_U32 nBitRate;
    OMX_U32 nSampleRate;
    OMX_U32 nBlocks;
    OMX_U32 nSubbands;
    OMX_U32 nBitPool;
    OMX_BOOL bEnableBitrate;
    OMX_AUDIO_CHANNELMODETYPE eChannelMode;
    OMX_AUDIO_SBCALLOCMETHODTYPE eSBCAllocType;
} OMX_AUDIO_PARAM_SBCTYPE;
```

#### **4.1.15.14.1.14.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_SBCTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of audio channels.
- nBitRate is the bit rate of the encoded SBC audio. If the bit rate is variable or unknown, this parameter has the value 0.
- nSampleRate is the sample rate of the source data. If the sample rate is variable or unknown, this parameter has the value 0.
- nBlocks is the block length with which the stream has been encoded.
- nSubbands is the number of frequency subbands.
- nBitPool is the size of the bit allocation pool used for encoding the stream.
- bEnableBitrate is the Boolean value to use nBitRate or nBitPool.
- eChannelMode is the audio channel mode.
- eSBCAllocType is the enumeration of the adaptive bit allocation algorithm. [Table 4-16](#) shows the field names and values.

**Table 4-16: Adaptive Bit Allocation Algorithm Values**

Field Name	Description
OMX_AUDIO_SBCAllocMethodLoudness	Loudness allocation method
OMX_AUDIO_SBCAllocMethodSNR	Signal-to-noise ratio (SNR) allocation method

#### **4.1.15.24.1.14.2 Functionality**

This OMX\_AUDIO\_PARAM\_SBCTYPE structure configures the parameters of the SBC codec.

#### **4.1.164.1.15 OMX\_AUDIO\_PARAM\_ADPCMTYPE**

Adaptive Differential PCM (ADPCM) is a waveform coding generic algorithm. It can be implemented in many ways and with different rates.

The OMX\_AUDIO\_PARAM\_ADPCMTYPE structure is used to set or query the current ~~or default~~ settings for the ADPCM codec component using the OMX\_GetParameter function. It is also used to set the parameters of the ADPCM codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioAdpcm.

OMX\_AUDIO\_PARAM\_ADPCMTYPE is defined as follows.

```

typedef struct OMX_AUDIO_PARAM_ADPCMTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_U32 nBitsPerSample;
    OMX_U32 nSampleRate;
    OMX_U32 nBlockSize;
} OMX_AUDIO_PARAM_ADPCMTYPE;

```

#### **4.1.16.14.1.15.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_ADPCM TYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo).
- nBitsPerSample is the number of bits per sample of audio.
- nSampleRate is the sampling rate of the source data. Use the value 0 for variable or unknown sampling rate.
- nBlockSize is the ADPCM block coding size.

#### **4.1.16.24.1.15.2 Functionality**

The OMX\_AUDIO\_PARAM\_ADPCM TYPE structure sets the parameters of a generic ADPCM codec.

#### **4.1.174.1.16 OMX\_AUDIO\_PARAM\_G723TYPE**

ITU G.723.1 is a standard speech codec that has two rates, 5.3 and 6.3 kbps, and is used in video telephony. The input sampling rate is 8 kHz.

The OMX\_AUDIO\_PARAM\_G723TYPE structure is used to set or query the current or default settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioG723.

OMX\_AUDIO\_PARAM\_G723TYPE is defined as follows.

```

typedef struct OMX_AUDIO_PARAM_G723TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_BOOL bDTX;
    OMX_AUDIO_G723RATE eBitRate;
} OMX_AUDIO_PARAM_G723TYPE;

```

```

    OMX_BOOL bHiPassFilter;
    OMX_BOOL bPostFilter;
} OMX_AUDIO_PARAM_G723TYPE;

```

#### 4.1.17.14.1.16.1 Parameter Definitions

The parameters of OMX\_AUDIO\_PARAM\_G723TYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo).
- bDTX enables Discontinuous Transmission according to Annex A of the standard.
- eBitRate is the bit rate of the encoded speech. [Table 4-17](#) identifies bit rate values.

**Table 4-17: G.723 Bit Rate Values**

Field Name	Description
OMX_AUDIO_G723ModeUnused	Rate unused or unknown
OMX_AUDIO_G723ModeLow	5.3 kbps
OMX_AUDIO_G723ModeHigh	6.3 kbps

- bHiPassFilter enables high-pass filter preprocessing in the encoder.
- bPostFilter enables post filter processing.

#### 4.1.17.24.1.16.2 Functionality

The OMX\_AUDIO\_PARAM\_G723TYPE structure sets the parameters of the ITU-G.723.1 codec.

#### 4.1.184.1.17 OMX\_AUDIO\_PARAM\_G726TYPE

ITU G.726 is a standard ADPCM waveform codec having four rates. The rate of 32 kbps is the most used rate and identical to an older standard, ITU G.721. The input sampling rate is 8 kHz.

The OMX\_AUDIO\_PARAM\_G726TYPE structure is used to set or query the current or default settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioG726.

OMX\_AUDIO\_PARAM\_G726TYPE is defined as follows.

```

typedef struct OMX_AUDIO_PARAM_G726TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
}

```

```

    OMX_U32 nChannels;
    OMX_AUDIO_G726MODE eG726Mode;
} OMX_AUDIO_PARAM_G726TYPE;

```

#### **4.1.18.14.1.17.1 Parameter Definitions**

The parameters of OMX\_AUDIO\_PARAM\_G726TYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo).
- eG726Mode is the bit rate of the encoded speech. [Table 4-18](#) identifies the bit rate values.

**Table 4-18: G.726 Bit Rate Values**

Field Name	Description
OMX_AUDIO_G726ModeUnused	Rate unused or unknown
OMX_AUDIO_G726Mode16	16 kbps
OMX_AUDIO_G726Mode24	24 kbps
OMX_AUDIO_G726Mode32	32 kbps (equals G.721)
OMX_AUDIO_G726Mode40	40 kbps

#### **4.1.18.24.1.17.2 Functionality**

The OMX\_AUDIO\_PARAM\_G726TYPE structure sets the parameters of the ITU-G.726 codec.

#### **4.1.194.1.18 OMX\_AUDIO\_PARAM\_G729TYPE**

ITU G.729 is a standard speech codec with a coding rate of 8 kbps that is used in various applications. The input sampling rate is 8 kHz. A bit-compatible, low-complexity version is called G.729 appendix A (or G.729A). Support for DTX is described in annex B of the G.729 standard.

The OMX\_AUDIO\_PARAM\_G729TYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioG729.

OMX\_AUDIO\_PARAM\_G729TYPE is defined as follows.

```

typedef struct OMX_AUDIO_PARAM_G729TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_BOOL bDTX;
}

```

```

    OMX_AUDIO_G729TYPE eBitType;
} OMX_AUDIO_PARAM_G729TYPE;

```

#### **4.1.19.14.1.18.1 Parameter Definitions**

The parameters of OMX\_AUDIO\_PARAM\_G729TYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo).
- bDTX enables Discontinuous Transmission when Annex B of the standard is used.
- eBitType identifies the standard annexes used. [Table 4-19](#) identifies the standard annexes.

**Table 4-19: Standard Annexes**

Field Name	Description
OMX_AUDIO_G729	G.729 without annexes
OMX_AUDIO_G729A	G.729 with annex A
OMX_AUDIO_G729B	G.729 with annex B
OMX_AUDIO_G729AB	G.729 with annexes A and B

#### **4.1.19.24.1.18.2 Functionality**

The OMX\_AUDIO\_PARAM\_G729TYPE structure sets the parameters of the ITU-G.729 codec.

#### **4.1.204.1.19 OMX\_AUDIO\_PARAM\_AMRTYPE**

The Adaptive Multi-Rate coder is defined in 3GPP standards as having [three](#)[two](#) main versions:

- Narrow Band (AMR-NB), where the sampling rate is 8 kHz. It is defined in standards 26.07x and 26.09x. This version is used in cellular phones and other wireless devices mainly for speech conversation.
- Wide Band (AMR-WB), where the sampling rate is 16 kHz. It is defined in standards 26.17x and 26.19x, and in ITU G.722.2. This version is used in cellular phones and other wireless devices mainly for streaming and voice-over-IP (VoIP) communication.
- [Extended Wide Band \(AMR-WB+\). New features include extended audio bandwidth, support for stereophonic audio, and the option to operate on and switch between several internal sampling frequencies \(ISF\). AMR-WB+ decoder can output signals at a variety of sampling rates \(8kHz, 11.025kHz, 16kHz, 32kHz, 22.05kHz, 24kHz, 32kHz, 44.1kHz, 48kHz\). Note that the desired PCM output sampling rate in case of AMR-WB+ decoder should be specified as the output](#)

port PCM nSamplingRate parameter. The AMR-WB+ encoder should be notified of the PCM input sampling rate through the input port PCM nSamplingRate parameter. AMR-WB+ is defined in standards 3GPP TS 26.290 and in RFC 4352. The expected key application for AMR-WB+ is streaming.

The OMX\_AUDIO\_PARAM\_AMRTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioAmr.

OMX\_AUDIO\_PARAM\_AMRTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_AMRTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_U32 nBitRate;
    OMX_AUDIO_AMRBANDMODETYPE eAMRBandMode;
    OMX_AUDIO_AMRDTXMODETYPE eAMRDTXMode;
    OMX_AUDIO_AMRFRAMEFORMATTYPE eAMRFrameFormat;
    OMX_AUDIO_AMRISFINDEXTYPE eAMRISFIndex;
} OMX_AUDIO_PARAM_AMRTYPE;
```

#### 4.1.20.14.1.19.1 Parameter Definitions

The parameters of OMX\_AUDIO\_PARAM\_AMRTYPE are defined as follows.

- nPortIndex is the read only value containing the index of the portrepresents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo).
- nBitrate is the bit rate of the encoded AMR audio. This parameter is a read only parameter used to query the current bitrate of the audio. If the bit rate is variable or unknown, this parameter has the value 0.
- eAMRBandMode is the bit rate of the encoded speech. Table 4-20 shows the bit rate values. Note that bit rate values for AMR-WB+ entries OMX\_AUDIO\_AMRBandModeWBP16 - OMX\_AUDIO\_AMRBandModeWBP47 assume internal sampling frequency (ISF) of 25600 Hz.

Table 4-20: Adaptive Multi-Rate Bit Rate Values

Field Name	Description
OMX_AUDIO_AMRBandModeUnused	Rate unused or unknown
OMX_AUDIO_AMRBandModeNB0	4.75 kbps
OMX_AUDIO_AMRBandModeNB1	5.15 kbps
OMX_AUDIO_AMRBandModeNB2	5.9 kbps

<b>Field Name</b>	<b>Description</b>
OMX_AUDIO_AMRBandModeNB3	6.7 kbps
OMX_AUDIO_AMRBandModeNB4	7.4 kbps
OMX_AUDIO_AMRBandModeNB5	7.95 kbps
OMX_AUDIO_AMRBandModeNB6	10.2 kbps
OMX_AUDIO_AMRBandModeNB7	12.2 kbps
OMX_AUDIO_AMRBandModeWB0	6.6 kbps
OMX_AUDIO_AMRBandModeWB1	8.85 kbps
OMX_AUDIO_AMRBandModeWB2	12.65 kbps
OMX_AUDIO_AMRBandModeWB3	14.25 kbps
OMX_AUDIO_AMRBandModeWB4	15.85 kbps
OMX_AUDIO_AMRBandModeWB5	18.25 kbps
OMX_AUDIO_AMRBandModeWB6	19.85 kbps
OMX_AUDIO_AMRBandModeWB7	23.05 kbps
OMX_AUDIO_AMRBandModeWB8	23.85 kbps
<u>OMX_AUDIO_AMRBandModeWBP0</u>	<u>6.6 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP1</u>	<u>8.85 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP2</u>	<u>12.65 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP3</u>	<u>14.25 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP4</u>	<u>15.85 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP5</u>	<u>18.25 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP6</u>	<u>19.85 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP7</u>	<u>23.05 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP8</u>	<u>23.85 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP9</u>	<u>SID</u>
<u>OMX_AUDIO_AMRBandModeWBP10</u>	<u>13.6 kbps Mono</u>
<u>OMX_AUDIO_AMRBandModeWBP11</u>	<u>18 kbps Stereo</u>
<u>OMX_AUDIO_AMRBandModeWBP12</u>	<u>24 kbps Mono</u>
<u>OMX_AUDIO_AMRBandModeWBP13</u>	<u>24 kbps Stereo</u>
<u>OMX_AUDIO_AMRBandModeWBP14</u>	<u>FRAME ERASURE</u>
<u>OMX_AUDIO_AMRBandModeWBP15</u>	<u>NO DATA</u>
<u>OMX_AUDIO_AMRBandModeWBP16</u>	<u>10.4 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP17</u>	<u>12 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP18</u>	<u>13.6 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP19</u>	<u>15.2 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP20</u>	<u>16.8 kbps</u>
<u>OMX_AUDIO_AMRBandModeWBP21</u>	<u>19.2 kbps</u>

Field Name	Description
<a href="#">OMX_AUDIO_AMRBandModeWBP22</a>	<u>20.8 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP23</a>	<u>24 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP24</a>	<u>12.4 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP25</a>	<u>12.8 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP26</a>	<u>14 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP27</a>	<u>14.4 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP28</a>	<u>15.2 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP29</a>	<u>16 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP30</a>	<u>16.4 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP31</a>	<u>17.2 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP32</a>	<u>18 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP33</a>	<u>18.4 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP34</a>	<u>19.2 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP35</a>	<u>20 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP36</a>	<u>20.4 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP37</a>	<u>21.2 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP38</a>	<u>22.4 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP39</a>	<u>23.2 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP40</a>	<u>24 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP41</a>	<u>25.6 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP42</a>	<u>26 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP43</a>	<u>26.8 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP44</a>	<u>28.8 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP45</a>	<u>29.6 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP46</a>	<u>30 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeWBP47</a>	<u>32 kbps</u>
<a href="#">OMX_AUDIO_AMRBandModeAuto</a>	Allow encoder to select mode

- eAMRDTXMode identifies the AMR Discontinuous Transmission mode and voice activity detection (VAD) type. [Table 4-21](#) describes the modes and types.

**Table 4-21: Adaptive Multi-Rate Discontinuous Transmission Mode and VAD Type**

Field Name	Description
<a href="#">OMX_AUDIO_AMRDTXMode</a> <del>Off</del> <ins>Used</ins>	DTX <del>not used or unused</del>
<a href="#">OMX_AUDIO_AMRDTXModeOnVAD1</a>	Use Type 1 VAD
<a href="#">OMX_AUDIO_AMRDTXModeOnVAD2</a>	Use Type 2 VAD
<a href="#">OMX_AUDIO_AMRDTXModeOnAuto</a>	VAD type automatic

Field Name	Description
OMX_AUDIO_AMRDTXasEFR	DTX frames as EFR (3GPP 26.101, frame type equals 8,9,10)

- eAMRFrameFormat identifies the encoded frame format. [Table 4-22](#) shows the frame formats.

**Table 4-22: Encoded Frame Format**

Field Name	Description
OMX_AUDIO_AMRFrameFormatConformance	Standard test-sequence format (3GPP 26.074)
OMX_AUDIO_AMRFrameFormatIF1	Interface format 1 (NB- 3GPP 26.101, sec. 4 WB- 3GPP 26.201, sec. 4)
OMX_AUDIO_AMRFrameFormatIF2	Interface format 2 (NB- 3GPP 26.101, annex A WB- 3GPP 26.201, annex A)
OMX_AUDIO_AMRFrameFormatFSF	File Storage format (RFC <a href="#">48673267</a> , sec. 5)
OMX_AUDIO_AMRFrameFormatRTPPayloadFull	RTP payload format (RFC <a href="#">48673267</a> , sec. 4)
OMX_AUDIO_AMRFrameFormatITU	ITU frame format
<a href="#">OMX_AUDIO_AMRFrameFormatRTPPayloadConstrained</a>	<a href="#">RTP payload format</a> (RFC 4867, sec. 4.4)
<a href="#">OMX_AUDIO_AMRFrameFormatWBPlusTIF</a>	<a href="#">AMR-WB+ Transport Interface Format</a> (3GPP TS 26.290, sec. 8.2)
<a href="#">OMX_AUDIO_AMRFrameFormatWBPlusFSF</a>	<a href="#">AMR-WB+ File Storage Format</a> (3GPP TS 26.290, sec. 8.3)
<a href="#">OMX_AUDIO_AMRFrameFormatWBPlusRTPPayloadBasic</a>	<a href="#">RTP payload basic format</a> (RFC 4352, sec. 4.2 and 4.3.2.1)
<a href="#">OMX_AUDIO_AMRFrameFormatWBPlusRTPPayloadInterleaved</a>	<a href="#">RTP payload interleaved format</a> (RFC 4352, sec. 4.2 and 4.3.2.2)

[OMX\\_AUDIO\\_AMRFrameFormatRTPPayloadFull](#) format shall be used to specify AMR RTP payloads that do not satisfy one or more of the constraints that apply to [OMX\\_AUDIO\\_AMRFrameFormatRTPPayloadConstrained](#) format specified below.

OMX\_AUDIO\_AMRFrameFormatRTPPayloadConstrained format is reserved for the most commonly used AMR RTP payload format that satisfies all of the following constraints:

- RTP payload data is single-channel
- RTP payload data is in octet-aligned mode
- RTP payload data is not robust-sort-ordered
- RTP payload data is non-interleaved
- RTP payload data does not include frame CRCs

Furthermore, in case of

OMX\_AUDIO\_AMRFrameFormatRTPPayloadConstrained format, the payload of a compressed data buffer delivered to the AMR decoder component shall omit the RTP header and payload header, and shall consist of the payload table of contents (TOC) followed by the speech data in normal order, as described in sections 4.4.2, 4.4.3 and 4.4.4 of the RFC 4867 document.

OMX\_AUDIO\_AMRFrameFormatWBplusRTPPayloadBasic and OMX\_AUDIO\_AMRFrameFormatWBplusRTPPayloadInterleaved format - the payload of a compressed data buffer delivered to the AMR decoder component shall contain the payload header, table of contents (TOC) followed by audio data as described in sections 4.2, and 4.3 or RFC 4352 document.

- eAMRISFIndex identifies the internal sampling frequency (ISF) index for AMR-WB+ given in 3GPP TS 26.290 document. In case of encoder, when the decision about the value of ISF is left to the encoder component, the parameter is to be set to OMX\_AUDIO\_AMRISFIndexAuto. If the ISF is unknown or variable, the parameter has the value OMX\_AUDIO\_AMRISFIndexUnknown. This parameter shall be ignored for AMR-NB and AMR-WB audio. Table 4-23 describes the values of the parameter.

**Table 4-23: Adaptive Multi-Rate Extended Wide Band Internal Sampling Frequency Index**

Field Name	Description
<u>OMX_AUDIO_AMRISFIndex0</u>	N/A
<u>OMX_AUDIO_AMRISFIndex1</u>	12800 Hz
<u>OMX_AUDIO_AMRISFIndex2</u>	14400 Hz
<u>OMX_AUDIO_AMRISFIndex3</u>	16000 Hz
<u>OMX_AUDIO_AMRISFIndex4</u>	17067 Hz
<u>OMX_AUDIO_AMRISFIndex5</u>	19200 Hz
<u>OMX_AUDIO_AMRISFIndex6</u>	21333 Hz
<u>OMX_AUDIO_AMRISFIndex7</u>	24000 Hz
<u>OMX_AUDIO_AMRISFIndex8</u>	25600 Hz

<u>Field Name</u>	<u>Description</u>
<u>OMX_AUDIO_AMRISFIndex9</u>	<u>28800 Hz</u>
<u>OMX_AUDIO_AMRISFIndex10</u>	<u>32000 Hz</u>
<u>OMX_AUDIO_AMRISFIndex11</u>	<u>34133 Hz</u>
<u>OMX_AUDIO_AMRISFIndex12</u>	<u>36000 Hz</u>
<u>OMX_AUDIO_AMRISFIndex13</u>	<u>38400 Hz</u>
<u>OMX_AUDIO_AMRISFIndexAuto</u>	<u>Allow encoder to select ISF</u>
<u>OMX_AUDIO_AMRISFIndexUnknown</u>	<u>N/A</u>

#### 4.1.20.24.1.19.2     Functionality

The OMX\_AUDIO\_PARAM\_AMRTYPE structure sets the parameters of the AMR codec.

#### 4.1.214.1.20       OMX\_AUDIO\_PARAM\_GSMFRTYPE

The GSM Full-Rate codec is defined in ETSI standards 06.1x and 06.3x, which became 3GPP standards 26.01x and 26.03x.

The GSM Full-Rate coder is used in legacy GSM cellular phones. The sampling rate is 8 kHz. The encoded speech has a rate of 13 kbps, or 260 bits per frame of 20 milliseconds. The coding algorithm is RPE-LTP.

-The OMX\_AUDIO\_PARAM\_GSMFRTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioGsm\_FR.

OMX\_AUDIO\_PARAM\_GSMFRTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_GSMFRTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bDTX;
    OMX_BOOL bHiPassFilter ;
} OMX_AUDIO_PARAM_GSMFRTYPE;
```

#### 4.1.21.14.1.20.1   Parameter Definitions

The parameters for OMX\_AUDIO\_PARAM\_GSMFRTYPE as defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- bDTX enables Discontinuous Transmission (3GPP 46.031, 46.032).
- bHiPassFilter enables high-pass filter processing

#### **4.1.21.24.1.20.2     Functionality**

The OMX\_AUDIO\_PARAM\_GSMFRTYPE structure sets the parameters of the GSM Full-Rate codec.

#### **4.1.224.1.21       OMX\_AUDIO\_PARAM\_GSMEFRTYPE**

The GSM Enhanced Full-Rate codec is defined in ETSI standards 06.5x, 06.6x, and 06.8x; these standards became 3GPP standards 26.05x, 26.06x, and 26.08x.

The GSM Enhanced Full-Rate codec is used in GSM cellular phones. The sampling rate is 8 kHz. The encoded speech has a rate of 12.2 kbps, or 244 bits per frame of 20 milliseconds. Each coded frame is augmented by 16 error-protection bits that provide the complement of 260 bits, which is the same as the Full Rate codec. However this augmentation is performed outside of the speech coder. The coding algorithm is ACELP.

-The OMX\_AUDIO\_PARAM\_GSMEFRTYPE structure is used to set or query the current **or default** settings for the codec component using the OMX\_SetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_SetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioGsm\_EFR.

OMX\_AUDIO\_PARAM\_GSMEFRTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_GSMEFRTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bDTX;
    OMX_BOOL bHiPassFilter;
} OMX_AUDIO_PARAM_GSMEFRTYPE;
```

#### **4.1.22.14.1.21.1   Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_GSMEFRTYPE are defined as follows.

- nPortIndex **is the read-only value containing the index of the port represents the port that this structure applies to.**
- bDTX enables Discontinuous Transmission (3GPP 46.041, 46.042).
- bHiPassFilter enables High-Pass filter preprocessing in the encoder.

#### **4.1.22.24.1.21.2   Functionality**

The OMX\_AUDIO\_PARAM\_GSMEFRTYPE structure sets the parameters of the GSM Enhanced Full-Rate codec.

#### **4.1.234.1.22 OMX\_AUDIO\_PARAM\_GSMHRTYPE**

The GSM Half-Rate codec is defined in ETSI standards 06.2x and 06.4x; these standards became 3GPP standards 26.02x and 26.04x.

The GSM Half-Rate codec is used in GSM cellular phones. The sampling rate is 8 kHz. The encoded speech has a rate of 5.6 kbps, or 112 bits per frame of 20 milliseconds. The coding algorithm is VSELP.

The OMX\_AUDIO\_PARAM\_GSMHRTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioGsm\_HR.

OMX\_AUDIO\_PARAM\_GSMHRTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_GSMHRTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BOOL bDTX;  
    OMX_BOOL bHiPassFilter;  
} OMX_AUDIO_PARAM_GSMHRTYPE;
```

#### **4.1.23.14.1.22.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_GSMHRTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- bDTX enables Discontinuous Transmission (3GPP 46.041, 46.042).
- bHiPassFilter enables High-Pass filter preprocessing in the encoder.

#### **4.1.23.24.1.22.2 Functionality**

The OMX\_AUDIO\_PARAM\_GSMHRTYPE structure sets the parameters of the GSM Half-Rate codec.

#### **4.1.244.1.23 OMX\_AUDIO\_PARAM\_TDMAFRTYPE**

The TDMA Full-Rate codec is defined in the TIA/EIA-136-420 American cellular standard, also referred to as IS-136. It is a legacy codec used in the American cellular standard known as DAMPS.

The sampling rate is 8 kHz. The encoded speech has a rate of 7.95 kbps, or 159 bits per frame of 20 milliseconds. The coding algorithm is VSELP.

The OMX\_AUDIO\_PARAM\_TDMAFRTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter

function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioTdma\_FR. OMX\_AUDIO\_PARAM\_TDMAFRTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_TDMAFRTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_BOOL bDTX;
    OMX_BOOL bHiPassFilter;
} OMX_AUDIO_PARAM_TDMAFRTYPE;
```

#### **4.1.24.14.1.23.1 Parameter Definitions**

The parameters of OMX\_AUDIO\_PARAM\_TDMAFRTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the port represents the port that this structure applies to.~~
- nChannels is the number of audio channels.
- bDTX enables Discontinuous Transmission.
- bHiPassFilter enables High-Pass filter preprocessing in the encoder.

#### **4.1.24.24.1.23.2 Functionality**

The OMX\_AUDIO\_PARAM\_TDMAFRTYPE structure sets the parameters of the TDMA Full-Rate codec.

#### **4.1.254.1.24 OMX\_AUDIO\_PARAM\_TDMAEFRTYPE**

The TDMA Enhanced Full-Rate codec is defined in the TIA/EIA-136-410 American cellular standard, which is also referred to as IS-641, DAMPS-EFR. It is the codec used in the American cellular standard known as DAMPS.

The sampling rate is 8 kHz. The encoded speech has a rate of 7.4 kbps, or 148 bits per frame of 20 milliseconds. The coding algorithm is ACELP.

-The OMX\_AUDIO\_PARAM\_TDMAEFRTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioTdma\_EFR.

OMX\_AUDIO\_PARAM\_TDMAEFRTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_TDMAEFRTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
```

```

OMX_U32 nPortIndex;
OMX_U32 nChannels;
OMX_BOOL bDTX;
OMX_BOOL bHiPassFilter;
} OMX_AUDIO_PARAM_TDMAEFRTYPE;

```

#### **4.1.25.14.1.24.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_TDMAEFRTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of audio channels.
- bDTX enables Discontinuous Transmission.
- bHiPassFilter enables High-Pass filter preprocessing in the encoder.

#### **4.1.25.24.1.24.2 Functionality**

The OMX\_AUDIO\_PARAM\_TDMAEFRTYPE structure sets the parameters of the TDMA Enhanced Full-Rate codec.

#### **4.1.264.1.25 OMX\_AUDIO\_PARAM\_PDCFRTYPE**

The PDC Full-Rate codec is defined in ARIB standard RCR-27B. It is the legacy codec used in the Japanese cellular system.

The sampling rate is 8 kHz. The encoded speech has a rate of 6.7 kbps, or 134 bits per frame of 20 milliseconds. The coding algorithm is VSELP.

The OMX\_AUDIO\_PARAM\_PDCFRTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioPdc\_FR.

OMX\_AUDIO\_PARAM\_PDCFRTYPE is defined as follows.

```

typedef struct OMX_AUDIO_PARAM_PDCFRTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_BOOL bDTX;
    OMX_BOOL bHiPassFilter;
} OMX_AUDIO_PARAM_PDCFRTYPE;

```

#### **4.1.26.14.1.25.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_PDCFRTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nChannels is the number of audio channels.
- bDTX enables Discontinuous Transmission.
- bHiPassFilter enables High-Pass filter preprocessing in the encoder.

#### **4.1.26.24.1.25.2     Functionality**

The OMX\_AUDIO\_PARAM\_PDCFRTYPE structure sets the parameters of the PDC Full-Rate codec.

#### **4.1.274.1.26       OMX\_AUDIO\_PARAM\_PDCEFRTYPE**

The PDC Full-Rate codec is defined in ARIB standard RCR-27H. The codec is used in the Japanese cellular system.

The sampling rate is 8 kHz. The encoded speech has a rate of 6.7 kbps, or 134 bits per frame of 20 milliseconds. The coding algorithm is ACELP.

-The OMX\_AUDIO\_PARAM\_PDCEFRTYPE structure is used to set or query the current or default settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioPdc\_EFR.

OMX\_AUDIO\_PARAM\_PDCEFRTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_PDCEFRTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_BOOL bDTX;
    OMX_BOOL bHiPassFilter;
} OMX_AUDIO_PARAM_PDCEFRTYPE;
```

#### **4.1.27.14.1.26.1   Parameter Definitions**

The parameters of OMX\_AUDIO\_PARAM\_PDCEFRTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nChannels is the number of audio channels.
- bDTX enables Discontinuous Transmission.
- bHiPassFilter enables High-Pass filter preprocessing in the encoder.

#### **4.1.27.24.1.26.2     Functionality**

The OMX\_AUDIO\_PARAM\_PDCEFRTYPE structure sets the parameters of the PDC Enhanced Full-Rate codec.

#### **4.1.284.1.27       OMX\_AUDIO\_PARAM\_PDCHRTYPE**

The PDC Full-Rate codec is defined in ARIB standard RCR-27C. The codec is used in the Japanese cellular system.

The sampling rate is 8 kHz. The encoded speech has a rate of 3.45 kbps, or 138 bits per frame of 40 milliseconds. The coding algorithm is PSI-CELP.

-The OMX\_AUDIO\_PARAM\_PDCHRTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioPdc\_HR.

OMX\_AUDIO\_PARAM\_PDCHRTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_PDCHRTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_BOOL bDTX;
    OMX_BOOL bHiPassFilter;
} OMX_AUDIO_PARAM_PDCHRTYPE;
```

#### **4.1.28.14.1.27.1   Parameter Definitions**

The parameters of OMX\_AUDIO\_PARAM\_PDCHRTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- nChannels is the number of audio channels.
- bDTX enables Discontinuous Transmission.
- bHiPassFilter enables High-Pass filter preprocessing in the encoder.

#### **4.1.28.24.1.27.2   Functionality**

The OMX\_AUDIO\_PARAM\_PDCHRTYPE structure sets the parameters of the PDC Full-Rate codec.

#### **4.1.294.1.28 OMX\_AUDIO\_PARAM\_QCELP8TYPE**

The QCELP (lower rate) variable rate codec is defined in the TIA/EIA-96 standard. It is the legacy codec used in the CDMA cellular standard, mainly in Korea and North America.

The sampling rate is 8 kHz. The encoded speech has a maximal rate called Rate 1 of 8 kbps, or 160 bits per frame of 20 milliseconds. The codec can work on lower rates, namely Rates 1/2, 1/4, and 1/8, depending on the speech activity and channel capacity. Rate 1 adds 11 parity bits per frame, so its rate becomes 8.55 kbps.

The coding algorithm is QCELP.

-The OMX\_AUDIO\_PARAM\_QCELP8TYPE structure is used to set or query the current **or default** settings for the codec component using the OMX\_SetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_SetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioQcelp8.

OMX\_AUDIO\_PARAM\_QCELP8TYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_QCELP8TYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nChannels;  
    OMX_U32 nBitRate;  
    OMX_AUDIO_CDMARATETYPE eCDMARate;  
    OMX_U32 nMinBitRate;  
    OMX_U32 nMaxBitRate;  
} OMX_AUDIO_PARAM_QCELP8TYPE;
```

#### **4.1.29.14.1.28.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_QCELP8TYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of audio channels.
- nBitRate is the bit rate of the audio stream. If the bit rate is unknown, this parameter has the value 0.
- eCDMARate is the frame rate or type. [Table 4-24](#) shows the frame rate values.

**Table 4-24: QCELP8-CDMA Frame Rate Values**

Field Name	Description
OMX_AUDIO_CDMARateBlank	Blank frame
OMX_AUDIO_CDMARateFull	Rate 1
OMX_AUDIO_CDMARateHalf	Rate 1/2

Field Name	Description
OMX_AUDIO_CDMARateQuarter	Rate 1/4
OMX_AUDIO_CDMARateEighth	Rate 1/8
OMX_AUDIO_CDMARateErasure	Erasure frame (due to channel errors)

- nMinBitRate is the minimal restriction on the encoder for the current frame. The value is 1, 2, 3, or 4. The default value is 1.
- nMaxBitRate is the maximal restriction on the encoder for the current frame. The value is 1, 2, 3, or 4. This value shall be greater than or equal to the minimal rate. The default value is 4.

#### 4.1.29.24.1.28.2 Functionality

The OMX\_AUDIO\_PARAM\_QCELP8TYPE structure sets the parameters of the QCELP8 codec.

#### 4.1.304.1.29 OMX\_AUDIO\_PARAM\_QCELP13TYPE

The QCELP (high-rate) variable rate codec is defined in the TIA/EIA-733 standard. It is the codec that is used in the high-rate service option of CDMA cellular standard, mainly in Korea and North America.

The sampling rate is 8 kHz. The encoded speech has a maximal rate called Rate 1 of 13.3 kbps, or 266 bits per frame of 20 milliseconds. The codec can work on lower rates, namely Rates 1/2, 1/4, and 1/8, depending on the capacity of the speech activity channel.

The coding algorithm is QCELP.

The OMX\_AUDIO\_PARAM\_QCELP13TYPE structure is used to set or query the current **or default** settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioQcelp13.

OMX\_AUDIO\_PARAM\_QCELP13TYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_QCELP13TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_AUDIO_CDMARATETYPE eCDMARate;
    OMX_U32 nMinBitRate;
    OMX_U32 nMaxBitRate;
} OMX_AUDIO_PARAM_QCELP13TYPE;
```

#### 4.1.30.14.1.29.1 Parameter Definitions

The parameters for OMX\_AUDIO\_PARAM\_QCELP13TYPE are defined as follows.

- nPortIndex is the read only value containing the index of the portrepresents the port that this structure applies to.
- nChannels is the number of audio channels.
- eCDMARate is the frame rate or type. [Table 4-24](#) in section 4.1.28.1 shows the frame rate values.

**Table 4-23: QCELP13 Frame Rate Values**

Field Name	Description
<code>OMX_AUDIO_CDMARateBlank</code>	Blank frame
<code>OMX_AUDIO_CDMARateFull</code>	Rate 1
<code>OMX_AUDIO_CDMARateHalf</code>	Rate 1/2
<code>OMX_AUDIO_CDMARateQuarter</code>	Rate 1/4
<code>OMX_AUDIO_CDMARateEighth</code>	Rate 1/8
<code>OMX_AUDIO_CDMARateErasure</code>	Erasure frame (due to channel errors)

- nMinBitRate is the minimal restriction on the encoder for the current frame. The value is 1, 2, 3, or 4. The default value is 1.
- nMaxBitRate is the maximal restriction on the encoder for the current frame. The value is 1, 2, 3, or 4. The value shall be greater than or equal to the minimal rate. The default value is 4.

#### **4.1.30.24.1.29.2      Functionality**

The OMX\_AUDIO\_PARAM\_QCELP13TYPE structure sets the parameters of the QCELP13 codec.

#### **4.1.314.1.30      OMX\_AUDIO\_PARAM\_EVRCTYPE**

The Enhanced Variable Speech Coder is defined in the TIA/EIA-127 standard. It is the codec used in the CDMA cellular standard, mainly in Korea and North America.

The sampling rate is 8 kHz. The encoded speech has a maximal rate, called Rate 1, of 8.55 kbps, or 171 bits per frame of 20 milliseconds. The codec can work on lower rates, namely Rate 1/2 and 1/8, depending on the speech activity and the channel capacity.

The coding algorithm is RCELP.

-The OMX\_AUDIO\_PARAM\_EVRCTYPE structure is used to set or query the current ~~or~~ default settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioEvrc.

OMX\_AUDIO\_PARAM\_EVRCTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_EVRCTYPE {
    OMX_U32 nSize;
```

```

OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_U32 nChannels;
OMX_AUDIO_CDMARATETYPE eCDMARate;
OMX_BOOL bRATE_REDUCOn;
OMX_U32 nMinBitRate;
OMX_U32 nMaxBitRate;
OMX_BOOL bHiPassFilter;
OMX_BOOL bNoiseSuppressor;
OMX_BOOL bPostFilter;
} OMX_AUDIO_PARAM_EVRCTYPE;

```

#### 4.1.31.14.1.30.1 Parameter Definitions

The parameters for OMX\_AUDIO\_PARAM\_EVRCTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nChannels is the number of audio channels.
- eCDMARate is the frame rate or type. [Table 4-24](#) in section 4.1.28.1 shows the frame rate values.

**Table 4-24: Enhanced Variable Speech Frame Rate Values**

Field Name	Description
<code>OMX_AUDIO_CDMARateBlank</code>	Blank frame
<code>OMX_AUDIO_CDMARateFull</code>	Rate 1
<code>OMX_AUDIO_CDMARateHalf</code>	Rate ½
<code>OMX_AUDIO_CDMARateEighth</code>	Rate 1/8
<code>OMX_AUDIO_CDMARateErasure</code>	Erasure frame (due to channel errors)

- bRATE\_REDUCOn specifies if rate reduction is required
- nMinBitRate is the minimal restriction on the encoder for the current frame. The value is 1, 3, or 4. The default value is 1.
- nMaxBitRate is the maximal restriction on the encoder for the current frame. The value is 1, 3, or 4. The value shall be greater than or equal to the minimal rate. The default value is 4.
- bHiPassFilter enables high-pass filter processing.
- bNoiseSuppressor enables the encoder's noise suppressor preprocessing as a part of the encoder.
- bPostFilter enables post filter processing.

#### **4.1.31.24.1.30.2     Functionality**

The OMX\_AUDIO\_PARAM\_EVRCTYPE structure sets the parameters of the Enhanced Variable Speech Coder (EVRC) speech codec.

#### **4.1.324.1.31       OMX\_AUDIO\_PARAM\_SMVTYPE**

The Selectable Mode Vocoder (SMV) is defined in 3GPP2 standard C.S0030-2. It is the codec used in the CDMA2000 cellular standard.

The sampling rate is 8 kHz. The encoded speech has a maximal rate, called Rate 1, of 8.55 kbps, or 171 bits per frame of 20 milliseconds. It can work on lower rates, namely Rates 1/2, 1/4, and 1/8, depending on the speech activity and the channel capacity.

The coding algorithm is eX-CELP.

-The OMX\_AUDIO\_PARAM\_SMVTYPE structure is used to set or query the current ~~or default~~ settings for the codec component using the OMX\_GetParameter function. It is also used to set the parameters of the codec component using the OMX\_SetParameter function. When calling either the OMX\_GetParameter or the OMX\_SetParameter functions, the index specified for this structure is OMX\_IndexParamAudioSmv.

OMX\_AUDIO\_PARAM\_SMVTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_SMVTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
    OMX_AUDIO_CDMARATETYPE eCDMARate;
    OMX_BOOL bRATE_REDUCOn;
    OMX_U32 nMinBitRate;
    OMX_U32 nMaxBitRate;
    OMX_BOOL bHiPassFilter;
    OMX_BOOL bNoiseSuppressor;
    OMX_BOOL bPostFilter;
} OMX_AUDIO_PARAM_SMVTYPE;
```

#### **4.1.32.14.1.31.1   Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_SMVTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the port represents the port that this structure applies to.~~
- nChannels is the number of audio channels.
- eCDMARate is the frame rate or type. [Table 4-24 in section 4.1.28.1](#) identifies the frame rate values.

**Table 4-25: Selectable Mode Vocoder Frame Rate Values**

Field Name	Description
OMX_AUDIO_CDMARateBlank	Blank frame

Field Name	Description
OMX_AUDIO_CDMARateFull	Rate 1
OMX_AUDIO_CDMARateHalf	Rate ½
OMX_AUDIO_CDMARateEighth	Rate 1/8
OMX_AUDIO_CDMARateErasure	Erasure frame (due to channel errors)

- `bRATE_REDUCOn` specifies if rate reduction is required
- `nMinBitRate` is the minimal restriction on the encoder for the current frame. The value is 1, 3, or 4. The default value is 1.
- `nMaxBitRate` is the maximal restriction on the encoder for current frame. The value is 1, 3, or 4. The value shall be greater than or equal to the minimal rate. The default value is 4.
- `bHiPassFilter` enables high-pass filter processing.
- `bNoiseSuppressor` enables the encoder's noise suppressor preprocessing as a part of the encoder.
- `bPostFilter` enables post filter processing.

#### **4.1.32.24.1.31.2     Functionality**

The `OMX_AUDIO_PARAM_SMVTYPE` structure sets the parameters of the Selectable Mode Vocoder codec.

#### **4.1.334.1.32       OMX\_AUDIO\_PARAM\_MIDITYPE**

The `OMX_AUDIO_PARAM_MIDITYPE` structure is used to set or query the initial basic parameters of the MIDI engine. The parameters define the number of output channels of PCM audio, the maximum polyphony that the device supports, and whether the default soundbank is loaded at initialization.

`OMX_AUDIO_PARAM_MIDITYPE` is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_MIDITYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nFileSize;
    OMX_BU32 sMaxPolyphony;
    OMX_BOOL bLoadDefaultSound;
    OMX_AUDIO_MIDIFORMATTYPE eMidiFormat;
} OMX_AUDIO_PARAM_MIDITYPE;
```

#### **4.1.33.14.1.32.1   Parameter Definitions**

The parameters for `OMX_AUDIO_PARAM_MIDITYPE` are defined as follows.

- `nPortIndex` is the read only value containing the index of the port represents the port that this structure applies to.
- `nFileSize` is the size of the MIDI file data in bytes. This field shall be specified by the IL client or the component configuring this port before data is accepted.
- `sMaxPolyphony` specifies the range of simultaneous polyphonic voices that are supported. Since this parameter is of type `OMX_BU32` (a bounded, unsigned 32-bit integer; see `OMX_Types.h`), it allows the querying and setting of minimum, nominal, and maximum values. A value of zero indicates that the default polyphony of the device is used.
- `bLoadDefaultSound` is a Boolean value that indicates whether the default soundbank is it to be loaded at initialization.
- `eMidiFormat` is an enumeration for the format of the MIDI file. Table 4-26 shows the MIDI file format.

**Table 4-26: MIDI File Format**

Field Name	Description
<code>OMX_AUDIO_MIDIFormatUnknown</code>	MIDI format is unknown, or not used or not required.
<code>OMX_AUDIO_MIDIFormatSMF0</code>	Standard MIDI File format 0
<code>OMX_AUDIO_MIDIFormatSMF1</code>	Standard MIDI File format 1
<code>OMX_AUDIO_MIDIFormatSMF2</code>	Standard MIDI File format 2
<code>OMX_AUDIO_MIDIFormatSPMIDI</code>	SP-MIDI
<code>OMX_AUDIO_MIDIFormatXMF0</code>	XMF type 0
<code>OMX_AUDIO_MIDIFormatXMF1</code>	XMF type 1
<code>OMX_AUDIO_MIDIFormatMobileXMF</code>	Mobile XMF (XMF type 2)
<code>OMX_AUDIO_MIDIFormatMax</code>	Allowance for expansion in the number of MIDI file formats

#### **4.1.344.1.33      OMX\_AUDIO\_PARAM\_MIDILOADUSERSONGTYPE**

The `OMX_AUDIO_PARAM_MIDILOADUSERSONGTYPE` structure is used to set or query the parameters required for loading and unloading user-specified MIDI downloadable soundbanks (DLS). This structure contains a major exception to the memory rules used in OpenMAX IL: It includes a pointer to data, namely the DLS, which is outside the structure. This is because DLS soundbanks can grow to upwards of 400 kB in some cases. Without this exception, the implementations would be forced to make redundant copies of these large soundbanks, wasting valuable system resources.

`OMX_AUDIO_PARAM_MIDILOADUSERSONGTYPE` is defined as follows.

```

typedef struct OMX_AUDIO_PARAM_MIDILOADUSERSONDTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nDLSIndex;
    OMX_U32 nDLSSize;
    OMX_PTR pDLSData;
    OMX_AUDIO_MIDISOUNDBANKTYPE eMidiSoundBank;
    OMX_AUDIO_MIDISOUNDBANKLAYOUTTYPE eMidiSoundBankLayout;
} OMX_AUDIO_PARAM_MIDILOADUSERSONDTYPE;

```

#### 4.1.34.14.1.33.1 Parameter Definitions

The parameters for OMX\_AUDIO\_PARAM\_MIDILOADUSERSONDTYPE are defined as follows.

- nPortIndex [is the read-only value containing the index of the port represents the port that this structure applies to.](#)
- nDLSIndex is the DLS file index to be loaded.
- nDLSSize is the size of the DLS in bytes.
- pDLSData is the pointer to the DLS file data.
- eMidiSoundBank is an enumeration for the various types of MIDI DLS soundbanks. Table 4-27 identifies the MIDI soundbanks.

**Table 4-27: MIDI Soundbanks**

Field Name	Description
OMX_AUDIO_MIDISoundBankUnused	Unused/unknown soundbank type
OMX_AUDIO_MIDISoundBankDLS1	DLS 1
OMX_AUDIO_MIDISoundBankDLS2	DLS 2
OMX_AUDIO_MIDISoundBankMobileDLSBase	Mobile DLS, using the base functionality
OMX_AUDIO_MIDISoundBankMobile_DLSPplusOptions	Mobile DLS, using the specification-defined optional feature set
<a href="#">OMX_AUDIO_MIDISoundBankMax</a>	<a href="#">Allowance for expansion in the number of soundbank types</a>

- eMidiSoundBankLayout is an enumeration for the various layouts of MIDI DLS soundbanks. Bank layout describes how the bank most significant bit (MSB) and least significant bit (LSB) are used in the DLS instrument definitions soundbank Table 4-28 shows the MIDI soundbank layouts.

**Table 4-28: MIDI Soundbank Layouts**

Field Name	Description
OMX_AUDIO_MIDISoundBankLayoutUnused	Unknown/unused soundbank layout type.
OMX_AUDIO_MIDISoundBankLayoutGM	GS layout based on bank MSB 0x00.

Field Name	Description
OMX_AUDIO_MIDI_SoundBankLayout_GM2	General MIDI 2 layout using MSB 0x78/0x79, LSB 0x00.
OMX_AUDIO_MIDI_SoundBankLayout_User	Does not conform to any bank numbering standards.
OMX_AUDIO_MIDI_SoundBankLayout_Max	<i>Allowance for expansion in the number of soundbank layout types.</i>

#### 4.1.35.4.1.34 OMX\_AUDIO\_CONFIG\_MIDIIMMEDIATEEVENTTYPE

The OMX\_AUDIO\_CONFIG\_MIDIIMMEDIATEEVENTTYPE structure is used to set the parameters for live MIDI events and Maximum Instantaneous Polyphony (MIP) messages, which are part of the SP-MIDI standard. The

OMX\_AUDIO\_CONFIG\_MIDIIMMEDIATEEVENTTYPE structure does not specify the format of MIDI events or MIP messages; it simply provides an array for the MIDI events or the MIP message buffer. The MIDI engine can parse this array and process it appropriately.

OMX\_AUDIO\_CONFIG\_MIDIIMMEDIATEEVENTTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_MIDIIMMEDIATEEVENTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nMidiEventSize;
    OMX_U8 nMidiEvents[1];
} OMX_AUDIO_CONFIG_MIDIIMMEDIATEEVENTTYPE;
```

#### 4.1.35.14.1.34.1 Parameter Definitions

The parameters for OMX\_AUDIO\_CONFIG\_MIDIIMMEDIATEEVENTTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nMidiEventSize is the size of the immediate MIDI events or MIP message in bytes.
- nMidiEvents is the MIDI event array to be rendered immediately, or an array for the MIP message buffer, where the size is indicated by nMidiEventSize.

#### 4.1.35.24.1.34.2 Post-processing Conditions

The live MIDI event array is rendered by the MIDI engine, or the MIP message contained in the buffer is processed.

#### **4.1.364.1.35 OMX\_AUDIO\_CONFIG\_MIDISOUNDBANKPROGRAMTYPE**

The OMX\_AUDIO\_CONFIG\_MIDISOUNDBANKPROGRAMTYPE structure is used to query and set the parameters for soundbank/program pairs in a given MIDI channel. It will be called once for each of the 16 MIDI channels. Note that the entire MIDI stream goes to a single port. One-to-one mapping does not occur between ports and MIDI channels.

OMX\_AUDIO\_CONFIG\_MIDISOUNDBANKPROGRAMTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_MIDISOUNDBANKPROGRAMTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nChannel;  
    OMX_U16 nIDProgram;  
    OMX_U16 nIDSoundBank;  
    OMX_U32 nUserSoundBankIndex;  
} OMX_AUDIO_CONFIG_MIDISOUNDBANKPROGRAMTYPE;
```

#### **4.1.36.14.1.35.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_MIDISOUNDBANKPROGRAMTYPE are defined as follows.

- nPortIndex is the read only value containing the index of the port represents the port that this structure applies to.
- nChannel refers to a MIDI channel. Valid channel values are 1 to 16.
- nIDProgram refers to a MIDI program. Valid program ID range is 1 to 128.
- nIDSoundBank is the soundbank ID.
- nUserSoundBankIndex is the user soundbank index. The index makes access to soundbanks easier if multiple banks are present.

#### **4.1.36.24.1.35.2 Post-processing Conditions**

The specified MIDI channel has a soundbank and program associated with it.

#### **4.1.374.1.36 OMX\_AUDIO\_CONFIG\_MIDICONTROLTYPE**

The OMX\_AUDIO\_CONFIG\_MIDICONTROLTYPE structure is used to query and set the parameters for controlling the rate and the looping (repeated playback) of MIDI playback.

OMX\_AUDIO\_CONFIG\_MIDICONTROLTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_MIDICONTROLTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BS32 sPitchTransposition;
```

```

OMX_BU32 sPlayBackRate;
OMX_BU32 sTempo ;
OMX_U32 nMaxPolyphony;
OMX_U32 nNumRepeat;
OMX_U32 nStopTime;
OMX_U16 nChannelMuteMask;
OMX_U16 nChannelSoloMask;
OMX_U32 nTrack0031MuteMask;
OMX_U32 nTrack3263MuteMask;
OMX_U32 nTrack0031SoloMask;
OMX_U32 nTrack3263SoloMask;
} OMX_AUDIO_CONFIG_MIDICONTROLTYPE;

```

#### 4.1.37.14.1.36.1 Parameter Definitions

The parameters for OMX\_AUDIO\_CONFIG\_MIDICONTROLTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- sPitchTransposition is the pitch transposition in semitones, stored as signed Q21.10 format, based on the Java MMAPI (JSR-135) requirement. As it is a bounded value type (OMX\_BS32), it allows the querying and setting of a range of values, including minimum, actual, and maximum.
- sPlayBackRate is the relative playback rate, stored as an unsigned Q154.17 fixed-point number based on the JSR-135 requirement. As it is a bounded value type (OMX\_BU32), it allows the querying and setting of a range of values, including minimum, actual, and maximum.
- sTempo is the tempo in beats per minute (BPM), stored as an unsigned Q22.10 fixed-point number based on the JSR-135 requirement. As it is a bounded value type (OMX\_BS32BU32), it allows the querying and setting of a range of values, including minimum, actual, and maximum.
- nMaxPolyphony specifies the maximum number of simultaneous polyphonic voices, which is the maximum run-time polyphony. A value of zero indicates that the default polyphony of the device is used.
- nNumRepeat specifies the number of times to repeat the playback.
- nStopTime is the time in milliseconds to indicate when playback will stop automatically. This value is set to zero if not used.
- nChannelMuteMask is a 16-bit mask for channel mute status.
- nChannelSoloMask is a 16-bit mask for channel solo status.
- nTrack0031MuteMask is a 32-bit mask for track mute status for tracks 0-31.
- nTrack3263MuteMask is a 32-bit mask for track mute status for tracks 32-63.
- nTrack0031SoloMask is a 32-bit mask for track solo status for tracks 0-31.

- nTrack3263SoloMask is a 32-bit mask for track mute status for tracks 32-63.

#### **4.1.37.24.1.36.2 Post-processing Conditions**

In case of a OMX\_SetConfig call using the OMX\_AUDIO\_CONFIG\_MIDICONTROLTYPE structure, the parameters required to control MIDI playback are set. In case of a OMX\_GetConfig call using the OMX\_AUDIO\_CONFIG\_MIDICONTROLTYPE structure, the MIDI IL client can determine the parameters controlling MIDI playback.

#### **4.1.384.1.37 OMX\_AUDIO\_CONFIG\_MIDISTATUSTYPE**

The OMX\_AUDIO\_CONFIG\_MIDISTATUSTYPE structure is used to query the current status of the MIDI playback. As such, it can be used only by an OMX\_GetConfig call. The OMX\_AUDIO\_CONFIG\_MIDISTATUSTYPE structure returns all of the parameters that characterize the current status of the MIDI engine.

OMX\_AUDIO\_CONFIG\_MIDISTATUSTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_MIDISTATUSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U16 nNumTracks;
    OMX_U32 nDuration;
    OMX_U32 nPosition;
    OMX_BOOL bVibra;
    OMX_U32 nNumMetaEvents;
    OMX_U32 nNumActiveVoices;
    OMX_AUDIO_MIDIPLAYBACKSTATETYPE eMIDIPlayBackState;
} OMX_AUDIO_CONFIG_MIDISTATUSTYPE;
```

#### **4.1.38.14.1.37.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_MIDISTATUSTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nNumTracks is a read-only field that identifies the number of MIDI tracks in the file. Note that this parameter will have a valid value only when the entire file has been parsed and buffered. An OMX\_GetConfig call issued before the entire file has been processed will not contain the correct number of MIDI tracks.
- nDuration is the length of the currently open MIDI resource in milliseconds. As with nNumTracks, this parameter will have a meaningful value only after the entire file has been buffered.
- nPosition is the current position in milliseconds of the MIDI resource being played.

- `bVibra` is a Boolean value that indicates if a vibra track exists in the file. This parameter will return a meaningful value only after the entire file has been buffered. The value returned when in the middle of the file cannot be relied upon.
- `nNumMetaEvents` is the total number of MIDI meta events in the currently open MIDI resource. This parameter will return a valid value only after the entire file is buffered. The value returned when in the middle of the file cannot be relied upon.
- `nNumActiveVoices` is the number of active voices in the currently playing MIDI resource, or the current polyphony level. This parameter may not return a meaningful value until the entire file is parsed and buffered.
- `eMIDIPlayBackState` is the enumeration for the MIDI playback state. Table 4-29 describes the playback states.

**Table 4-29: MIDI Playback States**

Field Name	Description
<code>OMX_AUDIO_MIDIPlayBackStateUnknown</code>	Unknown/unused MIDI playback state, or state does not map to one of the defined states.
<code>OMX_AUDIO_MIDIPlayBackStateClosed Engaged</code>	No MIDI resource is currently open. The MIDI engine is currently processing MIDI events.
<code>OMX_AUDIO_MIDIPlayBackStateParsing</code>	A MIDI resource is open and is being primed. The MIDI engine is currently processing MIDI events.
<code>OMX_AUDIO_MIDIPlayBackStateOpen Engaged</code>	A MIDI resource is open and primed but not playing. The MIDI engine is currently processing MIDI events. The transition to this state is only possible from the <code>OMX_AUDIO_MIDIPlayBackStatePlaying</code> state when the 'playback head' reaches the end of media data or the playback stops due to a stop time setting.
<code>OMX_AUDIO_MIDIPlayBackStatePlaying</code>	A MIDI resource is open and currently playing. The MIDI engine is currently processing MIDI events.
<code>OMX_AUDIO_MIDIPlayBackStatePlaying Partially</code>	Best-effort playback due to SP-MIDI/DLS resource constraints

Field Name	Description
OMX_AUDIO_MIDIPlayBackStatePlayingSilently	Due to system resource constraints and SP-MIDI content constraints, there is currently no audible MIDI content during playback. The situation may change if resources are freed later.
<b>OMX_AUDIO_MIDIPlayBackStateMax</b>	<b>Allowance for expansion in the number of playback states.</b>

#### 4.1.394.1.38 OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTTYPE

MIDI meta events are like audio metadata, except that they are interspersed with the MIDI content throughout the file and not localized in the header. As such, it is necessary to retrieve information about these meta-events from the engine as it encounters these meta events within the MIDI content. Component vendors are not required to enumerate all types of meta events; vendors can choose the meta events they want to support. Meta events are enumerated in the same order that they are detected in the MIDI file. Meta event data will always be provided as binary data, as it is present in the MIDI file.

The OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTTYPE structure is used to query the meta event, its track number, and the size of the meta event data using `OMX_GetConfig`. This allows the application to quickly determine meta events of interest. If the application requires the meta event data, the OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTDATATYPE structure, which is defined in section [4.1.39](#), needs to be used in a second `OMX_GetConfig` call.

OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_MIDIMETAEVENTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nIndex;
    OMX_U8 nMetaEventType;
    OMX_U32 nMetaEventSize;
    OMX_U32 nTrack;
    OMX_U32 nPosition;
} OMX_AUDIO_CONFIG_MIDIMETAEVENTTYPE;
```

#### 4.1.39.14.1.38.1 Parameter Definitions

The parameters for OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTTYPE are defined as follows.

- `nPortIndex` is the read-only value containing the index of the port represents the port that this structure applies to.
- `nIndex` is the index of the meta event. Meta events will be numbered 0 to N-1, where N is the number of meta events that the MIDI decoder reports.

- nMetaEventType is the meta event type. The values are 0-127.
- nMetaEventSize is the size of the meta event in bytes.
- nTrack is the track number for the meta event.
- nPosition is the position of the meta event in milliseconds.

#### **4.1.404.1.39 OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTDATATYPE**

The OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTDATATYPE structure is typically used by the IL client via an OMX\_GetConfig call to retrieve the meta event data, after the type, size and track number of the meta event have been determined by a previous OMX\_GetConfig call using the OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTTYPE structure defined in section [4.1.38 above](#). The IL client is responsible for sizing the structure appropriately so that it can hold the meta event data.

OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTDATATYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_MIDIMETAEVENTDATATYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nIndex;
    OMX_U32 nMetaEventSize;
    OMX_U8 nData[1];
} OMX_AUDIO_CONFIG_MIDIMETAEVENTDATATYPE;
```

#### **4.1.40.14.1.39.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_MIDIMETAEVENTDATATYPE are defined as follows.

- nPortIndex **is the read-only value containing the index of the port represents the port that this structure applies to.**
- nIndex is the index of the meta event. Meta events are numbered 0 to N-1, where N is the number of meta events that the MIDI decoder reports.
- nMetaEventSize is the size of the meta event in bytes.
- nData is an array of one or more bytes of meta data as indicated by the nMetaEventSize field.

#### **4.1.414.1.40 OMX\_AUDIO\_CONFIG\_VOLUMETYPE**

The OMX\_AUDIO\_CONFIG\_VOLUMETYPE structure is used to adjust the audio volume for a port.

OMX\_AUDIO\_CONFIG\_VOLUMETYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_VOLUMETYPE {
    OMX_U32 nSize;
```

```

OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_BOOL bLinear;
OMX_BS32 sVolume;
} OMX_AUDIO_CONFIG_VOLUMETYPE;

```

#### **4.1.41.14.1.40.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_VOLUMETYPE are defined as follows.

- nPortIndex: ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- bLinear is a Boolean to indicate if the volume is to be set on a linear (0-100) or a logarithmic scale (millibel, which is abbreviated mB). ~~This is a read-only parameter.~~
- sVolume is the linear volume setting in the range 0-100, or the logarithmic volume setting for this port. The values for volume are in millibel (abbreviated mB, where 1 millibel = 1/100 decibel) relative to a gain of 1 (i.e., the output is the same as the input level). Values are in mB from nMax (maximum volume) to nMin (minimum volume, typically negative). Since the volume is voltage and not a power, it takes a setting of -600 mB to decrease the volume by half. If a component cannot accurately set the volume to the requested value, it shall set the volume to the closest value below the requested value. When getting the volume setting, the current actual volume shall be returned.

#### **4.1.424.1.41 OMX\_AUDIO\_CONFIG\_CHANNELVOLUMETYPE**

The OMX\_AUDIO\_CONFIG\_CHANNELVOLUMETYPE structure is used to adjust the audio volume for a channel via the OMX\_IndexConfigAudioChannelVolume config.

OMX\_AUDIO\_CONFIG\_CHANNELVOLUMETYPE is defined as follows.

```

typedef struct OMX_AUDIO_CONFIG_CHANNELVOLUMETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannel;
    OMX_BOOL bLinear;
    OMX_BS32 sVolume;
    OMX_BOOL bIsMIDI;
} OMX_AUDIO_CONFIG_CHANNELVOLUMETYPE;

```

#### **4.1.42.14.1.41.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_CHANNELVOLUMETYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~

- nChannel is the channel to select in the range 0 to N-1. ~~using~~Use OMX\_ALL to apply ~~the~~ volume setting~~s~~ to all channels.
- bLinear is the volume to be set on a linear scale (0-100) or a logarithmic scale (mB).
- sVolume is the linear volume setting in the range 0-100 or the logarithmic volume setting for this port. The values for volume are in millibel (abbreviated mB, where 1 millibel = 1/100 dB) relative to a gain of 1 (i.e., the output is the same as the input level). Values are in mB from nMax (maximum volume) to nMin (minimum volume, typically negative). Since the volume is voltage and not a power, it takes a setting of -600 mB to decrease the volume by half. If a component cannot accurately set the volume to the requested value, it shall set the volume to the closest value below the requested value. When getting the volume setting, the current actual volume shall be returned.
- bIsMIDI is OMX\_TRUE if nChannel refers to a MIDI channel, or OMX\_FALSE otherwise.

#### **4.1.434.1.42 OMX\_AUDIO\_CONFIG\_BALANCETYPE**

The OMX\_AUDIO\_CONFIG\_BALANCETYPE structure defines the audio left-right balance adjustment for a port.

OMX\_AUDIO\_CONFIG\_BALANCETYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_BALANCETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nBalance;
} OMX_AUDIO_CONFIG_BALANCETYPE;
```

##### **4.1.43.14.1.42.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_BALANCETYPE are as follows.

- nPortIndex ~~is the read only value containing the index of the port~~represents the port that this structure applies to. Select the input port to set just that port's balance. Select the output port to adjust the master balance.
- nBalance is the balance setting for this port. The values are -100 to 100, where -100 indicates all left, and no right.

#### **4.1.444.1.43 OMX\_AUDIO\_CONFIG\_MUTETYPE**

The OMX\_AUDIO\_CONFIG\_MUTETYPE structure adjusts the audio mute for a port.

OMX\_AUDIO\_CONFIG\_MUTETYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_MUTETYPE {
    OMX_U32 nSize;
```

```
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_BOOL bMute;
} OMX_AUDIO_CONFIG_MUTETYPE;
```

#### **4.1.44.14.1.43.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_MUTETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to. Select the input port to set just that port's mute setting. Select the output port to adjust the master mute.
- bMute identifies whether the port is muted (OMX\_TRUE) or playing normally (OMX\_FALSE).

#### **4.1.454.1.44 OMX\_AUDIO\_CONFIG\_CHANNELMUTETYPE**

The OMX\_AUDIO\_CONFIG\_CHANNELMUTETYPE structure adjusts the audio mute for a channel.

OMX\_AUDIO\_CONFIG\_CHANNELMUTETYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_CHANNELMUTETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannel;
    OMX_BOOL bMute;
    OMX_BOOL bIsMIDI;
} OMX_AUDIO_CONFIG_CHANNELMUTETYPE;
```

#### **4.1.45.14.1.44.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_CHANNELMUTETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to. Select the input port to set just that port's mute setting. Select the output port to adjust the master mute.
- nChannel is the channel to select in the range 0 to N-1. Use OMX\_ALL to apply volume settingsthe audio mute setting to all channels.
- bMute identifies whether port is muted (OMX\_TRUE) or playing normally (OMX\_FALSE).
- bIsMIDI identifies whether the channel is a MIDI channel. The values are OMX\_TRUE if nChannel refers to a MIDI channel, OMX\_FALSE if otherwise.

#### **4.1.464.1.45 OMX\_AUDIO\_CONFIG\_LOUDNESSTYPE**

The OMX\_AUDIO\_CONFIG\_LOUDNESSTYPE structure is used to enable or disable the loudness audio effect, which boosts the bass and the high frequencies to compensate for the limited hearing range of humans at the extreme ends of the audio spectrum. The setting can be changed using the OMX\_SetConfig function. The current state can be queried using the OMX\_GetConfig function. When calling either OMX\_SetConfig or OMX\_GetConfig, the index specified for this structure is OMX\_IndexConfigAudioLoudness.

OMX\_AUDIO\_CONFIG\_LOUDNESSTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_LOUDNESSTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BOOL bLoudness;  
} OMX_AUDIO_CONFIG_LOUDNESSTYPE;
```

#### **4.1.46.14.1.45.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_LOUDNESSTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- bLoudness enables the loudness if set to OMX\_TRUE or disables the loudness effect if set to OMX\_FALSE.

#### **4.1.474.1.46 OMX\_AUDIO\_CONFIG\_BASSTYPE**

The OMX\_AUDIO\_CONFIG\_BASSTYPE structure is used to enable or disable the low-frequency level (bass) audio effect, and to set or query the current bass level. The setting can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudioBass.

OMX\_AUDIO\_CONFIG\_BASSTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_BASSTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BOOL bEnable;  
    OMX_S32 nBass;  
} OMX_AUDIO_CONFIG_BASSTYPE;
```

#### **4.1.47.14.1.46.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_BASSTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- bEnable enables the bass-level setting if set to OMX\_TRUE or disables the bass-level setting if set to OMX\_FALSE.
- nBass is the bass-level setting for the port, as a continuous value from -100 to 100. The value -100 means minimum bass level, zero means no change in level, and 100 represents the maximum low-frequency boost.

#### **4.1.484.1.47 OMX\_AUDIO\_CONFIG\_TREBLETYPE**

The OMX\_AUDIO\_CONFIG\_TREBLETYPE structure is used to enable or disable the high-frequency level (treble) audio effect, and to set or query the current level. The setting can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudioTreble.

OMX\_AUDIO\_CONFIG\_TREBLETYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_TREBLETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnable;
    OMX_S32 nTreble;
} OMX_AUDIO_CONFIG_TREBLETYPE;
```

#### **4.1.484.1.47.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_TREBLETYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- bEnable enables the treble level setting if set to OMX\_TRUE or disables the treble level setting if set to OMX\_FALSE.
- nTreble is the treble-level setting for the port, as a continuous value from -100 to 100. The value -100 means minimum high-frequency level, zero means no change in level, and 100 represents the maximum high-frequency boost.

#### **4.1.494.1.48 OMX\_AUDIO\_CONFIG\_EQUALIZERTYPE**

The OMX\_AUDIO\_CONFIG\_EQUALIZERTYPE structure is used to set or query the current parameters of the graphic equalizer (EQ) effect. The settings can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudioEqualizer.

An equalizer modifies the audio signal by frequency-dependent amplification or attenuation. A graphic EQ typically lets the user control the character of sound by controlling the levels of several fixed-frequency bands. The bands are characterized by their center and crossover frequencies.

In practice, the calling application or framework is often first interested in the number of bands that the EQ implementation supports. This number can be queried by a single call to OMX\_GetConfig with sBandIndex set to zero. The query results in the same data structure with the maximum value of sBandIndex filled with N-1, where N is the number of frequency bands. The same structure will also contain the frequency and level limits for the first band. Similar queries for the rest of the bands yield the information needed, for example, to construct a user interface for the equalizer.

OMX\_AUDIO\_CONFIG\_EQUALIZERTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_EQUALIZERTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnable;
    OMX_BU32 sBandIndex;
    OMX_BU32 sCenterFreq;
    OMX_BS32 sBandLevel;
} OMX_AUDIO_CONFIG_EQUALIZERTYPE;
```

#### **4.1.49.14.1.48.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_EQUALIZERTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- bEnable enables the EQ effect if set to OMX\_TRUE or disables the EQ effect if set to OMX\_FALSE.
- sBandIndex is the index of the band to be set or retrieved. The upper limit is N-1, where N is the number of bands. The lower limit is 0.
- sCenterFreq is the center frequencies in Hz. This is a read-only element and is used by the caller to determine the lower, center, and upper frequency of this band.
- sBandLevel is the band level in millibels.

#### **4.1.504.1.49 OMX\_AUDIO\_CONFIG\_STEREOWIDENINGTYPE**

The OMX\_AUDIO\_CONFIG\_STEREOWIDENINGTYPE structure is used to enable or disable the stereo widening audio effect, and to set or query the current strength of the effect. The setting can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudioStereoWidening.

Stereo widening is a special case of the “audio virtualizer” effect, and is designed to remove the inside-the-head effect in headphone listening, or to extend the stereo image beyond the physical loudspeaker span in loudspeaker reproduction.

`OMX_AUDIO_CONFIG_STEREOWIDENINGTYPE` is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_STEREOWIDENINGTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnable;
    OMX_AUDIO_STEREOWIDENINGTYPE eWideningType;
    OMX_U32 nStereoWidening;
} OMX_AUDIO_CONFIG_STEREOWIDENINGTYPE;
```

#### 4.1.50.14.1.49.1 Parameter Definitions

The parameters for `OMX_AUDIO_CONFIG_STEREOWIDENINGTYPE` are defined as follows.

- `nPortIndex` is the read-only value containing the index of the port represents the port that this structure applies to.
- `bEnable` enables the stereo widening effect if set to `OMX_TRUE` or disables the stereo widening effect if set to `OMX_FALSE`.
- `eWideningType` is the stereo widening processing type, as shown in Table 4-30.

**Table 4-30: Stereo Widening Processing Type**

Field Name	Description
<code>OMX_AUDIO_StereoWideningHeadphones</code>	Stereo widening for headphones.
<code>OMX_AUDIO_StereoWideningLoudspeakers</code>	Stereo widening for two closely spaced loudspeakers.
<code>OMX_AUDIO_StereoWideningMax</code>	Allowance for expansion in the number of stereo widening types.

- `nStereoWidening` is the stereo widening setting for the port, as a continuous value from 0 (minimum effect) to 100 (maximum effect). If the component can implement only a discrete set of presets (say, only on or off), it may round the value to a nearest available setting. When getting the setting, the exact current value shall be returned.

#### 4.1.514.1.50 `OMX_AUDIO_CONFIG_CHORUSTYPE`

The `OMX_AUDIO_CONFIG_CHORUSTYPE` structure is used to enable or disable the chorus audio effect, and to set or query the current parameters of the effect. The settings

can be changed using the `OMX_SetConfig` function, and the current state can be queried using the `OMX_GetConfig` function. When calling either function, the index specified for this structure is `OMX_IndexConfigAudioChorus`.

Chorus is an audio effect that presents a sound, such as a vocal track, as though it was performed by two or more singers simultaneously. The effect is produced by feeding the sound through one or more delay lines with time-variant lengths, and summing the delayed signals with the original, non-delayed sound. The length of each delay line is modulated by a low-frequency signal. Modulation waveform and stereo output details are implementation dependent.

`OMX_AUDIO_CONFIG_CHORUSTYPE` is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_CHORUSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnable;
    OMX_BU32 sDelay;
    OMX_BU32 sModulationRate;
    OMX_U32 nModulationDepth;
    OMX_BU32 nFeedback;
} OMX_AUDIO_CONFIG_CHORUSTYPE;
```

#### **4.1.51.14.1.50.1 Parameter Definitions**

The parameters for `OMX_AUDIO_CONFIG_CHORUSTYPE` are defined as follows.

- `nPortIndex` is the read-only value containing the index of the port represents the port that this structure applies to.
- `bEnable` enables the chorus effect if set to `OMX_TRUE` or disables the chorus effect if set to `OMX_FALSE`.
- `sDelay` is the average delay in milliseconds.
- `sModulationRate` is the rate of modulation in mHz.
- `nModulationDepth` is the depth of modulation as a percentage of delay zero-to-peak. The range of values is 0-100.
- `nFeedback` is the feedback from the chorus output to the input in percentage.

#### **4.1.524.1.51 OMX\_AUDIO\_CONFIG\_REVERBERATIONTYPE**

The `OMX_AUDIO_CONFIG_REVERBERATIONTYPE` structure is used to enable or disable the reverberation effect, and to set or query the current parameters of the effect. The settings can be changed using the `OMX_SetConfig` function, and the current state can be queried using the `OMX_GetConfig` function. When calling either function, the index specified for this structure is `OMX_IndexConfigAudioReverberation`.

The reverberation effect models the effect of a room (room response) to the sound. The room response is divided into three sections: direct path, early reflections, and late

reverberation. This division and the effect parameters are essentially the same as used in the Interactive 3D Audio Rendering Guidelines – Level 2.0 by the Interactive Audio Special Interest Group (IASIG) of the MIDI Manufacturers Association (MMA). For more information on this specification, see <http://www.iasig.org/pubs/3dl2v1a.pdf>.

OMX\_AUDIO\_CONFIG\_REVERBERATIONTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_REVERBERATIONTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BOOL bEnable;  
    OMX_BS32 sRoomLevel;  
    OMX_BS32 sRoomHighFreqLevel;  
    OMX_BS32 sReflectionsLevel;  
    OMX_BU32 sReflectionsDelay;  
    OMX_BS32 sReverbLevel;  
    OMX_BU32 sReverbDelay;  
    OMX_BU32 sDecayTime;  
    OMX_BU32 nDecayHighFreqRatio;  
    OMX_U32 nDensity;  
    OMX_U32 nDiffusion;  
    OMX_BU32 sReferenceHighFreq;  
} OMX_AUDIO_CONFIG_REVERBERATIONTYPE;
```

#### 4.1.52.14.1.51.1 Parameter Definitions

The parameters for OMX\_AUDIO\_CONFIG\_REVERBERATIONTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- bEnable enables the reverberation effect if set to OMX\_TRUE or disables the reverberation effect if set to OMX\_FALSE.
- sRoomLevel is the intensity level for the whole room effect, including both early reflections and late reverberation, in millibels.
- sRoomHighFreqLevel is the attenuation in millibels at high frequencies relative to the intensity at low frequencies.
- sReflectionsLevel is the intensity level of early reflections, which are relative to the room level value, in millibels.
- sReflectionsDelay is the time delay in milliseconds of the first reflection relative to the direct path.
- sReverbLevel is the intensity level in millibels of late reverberation relative to the room level.
- sReverbDelay is the time delay in milliseconds from the first early reflection to the beginning of the late reverberation section.

- `sDecayTime` is the late reverberation decay time in milliseconds at low frequencies, defined as the time needed for the reverberation to decay by 60 dB.
- `nDecayHighFreqRatio` is the ratio of high-frequency decay time relative to low-frequency decay time as percentage in the range 0–100.
- `nDensity` is the modal density in the late reverberation decay as a percentage. The range of values is 0-100.
- `nDiffusion` is the echo density in the late reverberation decay as a percentage. The range of values is 0-100.
- `sReferenceHighFreq` is the reference high frequency in Hertz. This is the frequency used as the reference for all of the high-frequency parameter settings.

#### **4.1.53.4.1.52 OMX\_AUDIO\_CONFIG\_ECHOCANCELLATIONTYPE**

The `OMX_AUDIO_CONFIG_ECHOCANCELLATIONTYPE` structure is used to enable or disable echo canceling, which removes undesired echo from speech or audio. The setting can be changed using the `OMX_SetConfig` function, and the current state can be queried using the `OMX_GetConfig` function. When calling either function, the index specified for this structure is `OMX_IndexConfigAudioEchoCancelation`.

`OMX_AUDIO_CONFIG_ECHOCANCELLATIONTYPE` is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_ECHOCANCELLATIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_AUDIO_ECHOCANTYPE eEchoCancelation;
} OMX_AUDIO_CONFIG_ECHOCANCELLATIONTYPE;
```

#### **4.1.53.14.1.52.1 Parameter Definitions**

The parameters for `OMX_AUDIO_CONFIG_ECHOCANCELLATIONTYPE` are defined as follows.

- `nPortIndex` is the read-only value containing the index of the port represents the port that this structure applies to.
- `eEchoCancelation` is the enumeration for enabling/disabling echo cancellation and selecting the mode, as shown in Table 4-31.

**Table 4-31: Echo Cancellation Values**

Field Name	Description
<code>OMX_AUDIO_EchoCanOff</code>	Echo cancellation is disabled.
<code>OMX_AUDIO_EchoCanNormal</code>	Echo cancellation normal operation; echo from handset plastics and face.

Field Name	Description
OMX_AUDIO_EchoCanHFree	Echo cancellation optimized for hands-free operation.
OMX_AUDIO_EchoCanCarKit	Echo cancellation optimized for car kit (longer echo).
<b>OMX_AUDIO_EchoCanMax</b>	<b>Allowance for expansion with additional types.</b>

#### **4.1.54.1.53 OMX\_AUDIO\_CONFIG\_NOISEREDUCTIONTYPE**

The OMX\_AUDIO\_CONFIG\_NOISEREDUCTIONTYPE structure is used to enable or disable noise reduction processing, which removes undesired noise from audio. The setting can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudioNoiseReduction.

OMX\_AUDIO\_CONFIG\_NOISEREDUCTIONTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_NOISEREDUCTIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bNoiseReduction;
} OMX_AUDIO_CONFIG_NOISEREDUCTIONTYPE;
```

#### **4.1.54.14.1.53.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_NOISEREDUCTIONTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- bNoiseReduction enables noise reduction processing if set to OMX\_TRUE or disables noise reduction processing if set to OMX\_FALSE.

#### **4.1.54 OMX\_AUDIO\_CONFIG\_3DOUTPUTTYPE**

The OMX\_AUDIO\_CONFIG\_3DOUTPUTTYPE structure is used to set or query the output type of the 3D processing. The setting can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudio3DOutput.

OMX\_AUDIO\_CONFIG\_3DOUTPUTTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_3DOUTPUTTYPE {
    OMX_U32 nSize;
```

```

OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_AUDIO_3DOUTPUTTYPE e3DOutputType;
} OMX_AUDIO_CONFIG_3DOUTPUTTYPE;

```

#### **4.1.54.1 Parameter Definitions**

The parameters for `OMX_AUDIO_CONFIG_3DOUTPUTTYPE` are defined as follows.

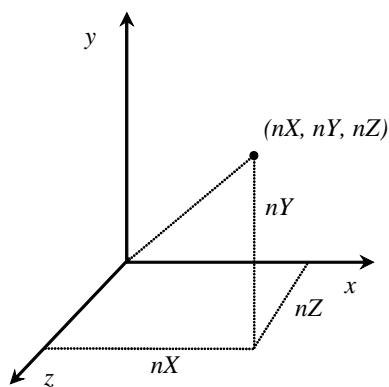
- `nPortIndex` represents the port that this structure applies to.
- `e3DOutputType` is the positional 3D audio processing type, as shown below

<b>Field Name</b>	<b>Description</b>
<code>OMX_AUDIO_3DOutputHeadphones</code>	Positional 3D audio for headphones.
<code>OMX_AUDIO_3DOutputLoudspeakers</code>	Positional 3D audio for two closely spaced loudspeakers.
<code>OMX_AUDIO_3DOutputMax</code>	Allowance for expansion in the number of positional 3D audio types.

#### **4.1.55 OMX AUDIO CONFIG 3DLOCATIONTYPE**

The `OMX_AUDIO_CONFIG_3DLOCATIONTYPE` structure is used to set the virtual location for the 3D sound source. The settings can be changed using the `OMX_SetConfig` function, and the current state can be queried using the `OMX_GetConfig` function. When calling either function, the index specified for this structure is `OMX_IndexConfigAudio3DLocation`.

The location is set by using right-handed coordinates relative to the listener. The listener is stationary: located in origin and pointing towards negative Z-axis up direction being the positive Y-axis.




---

`OMX_AUDIO_CONFIG_3DLOCATIONTYPE` is defined as follows.

```

typedef struct OMX_AUDIO_CONFIG_3DLOCATIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nX;
    OMX_S32 nY;
    OMX_S32 nZ;
} OMX_AUDIO_CONFIG_3DLOCATIONTYPE;

```

#### 4.1.55.1 Parameter Definitions

The parameters for OMX\_AUDIO\_CONFIG\_3DLOCATIONTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nX is the X coordinate in millimeters.
- nY is the Y coordinate in millimeters.
- nZ is the Z coordinate in millimeters.

#### 4.1.56 OMX AUDIO PARAM 3DDOPPLERMODETYPE

The OMX\_AUDIO\_PARAM\_3DDOPPLERMODETYPE structure is used to switch the Doppler effect on and off. The settings can be changed using the OMX\_SetParameter function, and the current state can be queried using the OMX\_GetParameter function. When calling either function, the index specified for this structure is OMX\_IndexParamAudio3DDopplerMode.

OMX\_AUDIO\_PARAM\_3DDOPPLERMODETYPE is defined as follows.

```

typedef struct OMX_AUDIO_PARAM_3DDOPPLERMODETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnabled;
} OMX_AUDIO_PARAM_3DDOPPLERMODETYPE;

```

#### 4.1.56.1 Parameter Definitions

The parameters for OMX\_AUDIO\_PARAM\_3DDOPPLERMODETYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- bEnabled: if true, the Doppler effect for this port is enabled; if false, the Doppler effect for this port is disabled.

#### 4.1.57 OMX AUDIO CONFIG 3DDOPPLERSETTINGSTYPE

The OMX\_AUDIO\_CONFIG\_3DDOPPLERSETTINGSTYPE structure is used to set the Doppler behavior of the 3D sound source. The settings can be changed using the

OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudio3DDopplerSettings.

OMX\_AUDIO\_CONFIG\_3DDOPPLERSETTINGSTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_3DDOPPLERSETTINGSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nSoundSpeed;
    OMX_S32 nSourceVelocity;
    OMX_S32 nListenerVelocity;
} OMX_AUDIO_CONFIG_3DDOPPLERSETTINGSTYPE;
```

The Doppler coefficient is usually calculated as:

$$Doppler = \frac{nSoundSpeed + nListenerVelocity}{nSoundSpeed - nSourceVelocity}$$

#### 4.1.57.1 Parameter Definitions

The parameters for OMX\_AUDIO\_CONFIG\_3DDOPPLERSETTINGSTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nSoundSpeed defines the speed of sound in millimeters per second.
- nSourceVelocity defines the speed of the source in the direction of the listener in millimeters per second. This is usually calculated as the velocity vector of the sound source projected on the line between source and listener positions.
- nListenerVelocity defines the speed of the listener in the direction of the source in millimeters per second. This is usually calculated as the velocity vector of the listener projected on the line between source and listener positions.

#### 4.1.58 OMX AUDIO CONFIG 3DLEVELSTYPE

The OMX\_AUDIO\_CONFIG\_3DLEVELSTYPE structure is used to set the direct path and room levels for the 3D sound source. The settings can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudio3DLevels.

OMX\_AUDIO\_CONFIG\_3DLEVELSTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_3DLEVELSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BS32 sDirectLevel;
    OMX_BS32 sRoomLevel;
} OMX_AUDIO_CONFIG_3DLEVELSTYPE;
```

```
} OMX_AUDIO_CONFIG_3DLEVELSTYPE;
```

#### 4.1.58.1 Parameter Definitions

The parameters for `OMX_AUDIO_CONFIG_3DLEVELSTYPE` are defined as follows.

- `nPortIndex` represents the port that this structure applies to.
- `sDirectLevel` is the level for direct path signal in millibels.
- `sRoomLevel` is the level for room signal in millibels.

#### 4.1.59 OMX\_AUDIO\_CONFIG\_3DDISTANCEATTENUATIONTYPE

The `OMX_AUDIO_CONFIG_3DDISTANCEATTENUATIONTYPE` structure is used to set the distance attenuation behavior for the 3D sound source. The settings can be changed using the `OMX_SetConfig` function, and the current state can be queried using the `OMX_GetConfig` function. When calling either function, the index specified for this structure is `OMX_IndexConfigAudio3DDistanceAttenuation`.

`OMX_AUDIO_CONFIG_3DDISTANCEATTENUATIONTYPE` is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_3DDISTANCEATTENUATIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BS32 sMinDistance;
    OMX_BS32 sMaxDistance;
    OMX_BS32 sRollOffFactor;
    OMX_BS32 sRoomRollOffFactor;
    OMX_AUDIO_ROLLOFFMODELTYPE eRollOffModel;
    OMX_BOOL bMuteAfterMax;
} OMX_AUDIO_CONFIG_3DDISTANCEATTENUATIONTYPE;
```

#### 4.1.59.1 Parameter Definitions

The parameters for `OMX_AUDIO_CONFIG_3DDISTANCEATTENUATIONTYPE` are defined as follows.

- `nPortIndex` represents the port that this structure applies to.
- `sMinDistance` is the  $d_{min}$  in the formulae below.
- `sMaxDistance` is the  $d_{max}$  in the formulae below.
- `sRollOffFactor` is the  $rolloffFactor$  in thousandths in the formulae below when calculating the distance attenuation for the direct path signal output.
- `sRoomRollOffFactor` is the  $rolloffFactor$  in thousandths in the formulae below, but when calculating the distance attenuation for the room signal output.
- `eRollOffModel` is the roll-off model defined below.

- sMuteAfterMax is the *rollOffMaxDistanceMute* in the formulae below.

Two distance roll-off models are supported. The exponential distance rolloff model is defined as:

$$gain(d) = \begin{cases} 1 & \text{if } d < d_{\min} \\ 0 & \text{if } d \geq d_{\max} \text{ and } rollOffMaxDistanceMute = \text{true} \\ \left(\frac{d_{\min}}{d}\right)^{rolloffFactor} & \text{if } d \geq d_{\max} \text{ and } rollOffMaxDistanceMute = \text{false} \\ \left(\frac{d_{\min}}{d}\right)^{rolloffFactor} & \text{otherwise} \end{cases}$$

And, the linear distance rolloff model is defined as:

$$gain(d) = \begin{cases} 1 & \text{if } d < d_{\min} \\ 0 & \text{if } d \geq d_{\max} \text{ and } rollOffMaxDistanceMute = \text{true} \\ \max(0, 1 - rollOffFactor) & \text{if } d \geq d_{\max} \text{ and } rollOffMaxDistanceMute = \text{false} \\ \max\left(0, 1 - \left(rolloffFactor \times \frac{d - d_{\min}}{d_{\max} - d_{\min}}\right)\right) & \text{otherwise} \end{cases}$$

where d is the distance of the sound source from the origin (listener).

#### 4.1.60 OMX\_AUDIO\_CONFIG\_3DDIRECTIVITYSETTINGSTYPE

The OMX\_AUDIO\_CONFIG\_3DDIRECTIVITYSETTINGSTYPE structure is used to set the directivity behavior of the 3D sound source. The settings can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudio3DDirectivitySettings.

OMX\_AUDIO\_CONFIG\_3DDIRECTIVITYSETTINGSTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_3DDIRECTIVITYSETTINGSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BS32 sInnerAngle;
    OMX_BS32 sOuterAngle;
    OMX_BS32 sOuterLevel;
} OMX_AUDIO_CONFIG_3DDIRECTIVITYSETTINGSTYPE;
```

##### 4.1.60.1 Parameter Definitions

The parameters for OMX\_AUDIO\_CONFIG\_3DDIRECTIVITYSETTINGSTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- sInnerAngle defines the inner cone in millidegrees. Within the inner cone the source radiates at its “full” level.
- sOuterAngle defines the outer cone in millidegrees. Outside of the outer cone the source radiates at outerLevel level relative to the full level.
- sOuterLevel defines the outerLevel defined above in millibels.

#### **4.1.61 OMX\_AUDIO\_CONFIG\_3DDIRECTIVITYORIENTATIONTYPE**

The OMX\_AUDIO\_CONFIG\_3DDIRECTIVITYORIENTATIONTYPE structure is used to set the orientation of the directivity of the 3D sound source. The settings can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudio3DDirectivityOrientation.

OMX\_AUDIO\_CONFIG\_3DDIRECTIVITYORIENTATIONTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_3DDIRECTIVITYORIENTATIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nXFront;
    OMX_S32 nYFront;
    OMX_S32 nZFront;
} OMX_AUDIO_CONFIG_3DDIRECTIVITYORIENTATIONTYPE;
```

##### **4.1.61.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_3DDIRECTIVITYORIENTATIONTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nXFront is the X component of the front direction vector of the sound source.
- nYFront is the Y component of the front direction vector of the sound source.
- nZFront is the Z component of the front direction vector of the sound source.

#### **4.1.62 OMX\_AUDIO\_CONFIG\_3DMACROSCOPICORIENTATIONTYPE**

The OMX\_AUDIO\_CONFIG\_3DMACROSCOPICORIENTATIONTYPE structure is used to set the orientation of the macroscopicity of the 3D sound source. The settings can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudio3DMacroscopicOrientation.

OMX\_AUDIO\_CONFIG\_3DMACROSCOPICORIENTATIONTYPE is defined as follows.

```

typedef struct OMX_AUDIO_CONFIG_3DMACROSCOPICORIENTATIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nXFront;
    OMX_S32 nYFront;
    OMX_S32 nZFront;
    OMX_S32 nXAbove;
    OMX_S32 nYAbove;
    OMX_S32 nZAbove;
} OMX_AUDIO_CONFIG_3DMACROSCOPICORIENTATIONTYPE;

```

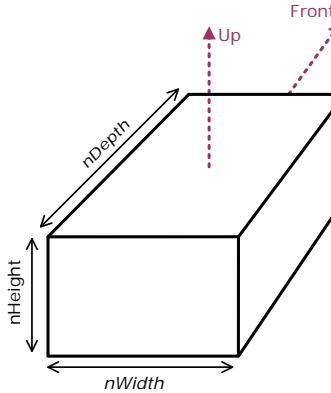
#### **4.1.62.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_CONFIG\_3DMACROSCOPICORIENTATIONTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nXFront is the X component of the front direction vector of the sound source.
- nYFront is the Y component of the front direction vector of the sound source.
- nZFront is the Z component of the front direction vector of the sound source.
- nXAbove is the X component of the above direction vector of the sound source.
- nYAbove is the Y component of the above direction vector of the sound source.
- nZAbove is the Z component of the above direction vector of the sound source.

#### **4.1.63 OMX\_AUDIO\_CONFIG\_3DMACROSCOPICSIZETYPE**

The OMX\_AUDIO\_CONFIG\_3DMACROSCOPICSIZETYPE structure is used to set the size of the macroscopicity of the 3D sound source. The settings can be changed using the OMX\_SetConfig function, and the current state can be queried using the OMX\_GetConfig function. When calling either function, the index specified for this structure is OMX\_IndexConfigAudio3DMacroscopicSize.



OMX\_AUDIO\_CONFIG\_3DMACROSCOPICSIZETYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_3DMACROSCOPICSIZETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nWidth;
    OMX_S32 nHeight;
    OMX_S32 nDepth;
} OMX_AUDIO_CONFIG_3DMACROSCOPICSIZETYPE;
```

#### 4.1.63.1 Parameter Definitions

The parameters for OMX\_AUDIO\_CONFIG\_3DMACROSCOPICSIZETYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nWidth is the width of the macroscopic sound source in millimeters.
- nHeight is the height of the macroscopic sound source in millimeters.
- nDepth is the depth of the macroscopic sound source in millimeters.

#### 4.1.64 OMX\_AUDIO\_CHANNELMAPPINGTYPE

The OMX\_AUDIO\_PARAM\_CHANNELMAPPINGTYPE structure is used to query the channel mapping information of the audio stream.

OMX\_AUDIO\_PARAM\_CHANNELMAPPINGTYPE is defined as follows.

```
typedef struct OMX_AUDIO_PARAM_CHANNELMAPPINGTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nChannels;
```

```
    OMX_AUDIO_CHANNELTYPE nChannelsMapping[OMX_AUDIO_MAXCHANNELS];
} OMX_AUDIO_PARAM_CHANNELMAPPINGTYPE;
```

#### **4.1.64.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_CHANNELMAPPINGTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nChannels is the number of channels of audio (mono, stereo, multi-channel).
- nChannelsMapping identifies the channel mappings available within the stream.

#### **4.1.65 OMX AUDIO SBCBITPOOLTYPE**

The OMX\_AUDIO\_SBCBITPOOLTYPE structure is used to set or query the SBC codec bit-pool parameter.

OMX\_AUDIO\_SBCBITPOOLTYPE is defined as follows.

```
typedef struct OMX_AUDIO_SBCBITPOOLTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nNewBitPool;
} OMX_AUDIO_SBCBITPOOLTYPE;
```

#### **4.1.65.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_CHANNELMAPPINGTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nNewBitPool is the size of the bit allocation pool used for encoding the stream.

#### **4.1.66 OMX AUDIO AMRMODETYPE**

The OMX\_AUDIO\_AMRMODETYPE structure is used to set or query the AMR codec mode and bitrate settings.

OMX\_AUDIO\_AMRMODETYPE is defined as follows.

```
typedef struct OMX_AUDIO_AMRMODETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nBitRate;
    OMX_AUDIO_AMRBANDMODETYPE eAMRBandMode ;
} OMX_AUDIO_AMRMODETYPE;
```

```
} OMX_AUDIO_AMRMODETYPE;
```

#### **4.1.66.1 Parameter Definitions**

The parameters for OMX\_AUDIO\_PARAM\_CHANNELMAPPINGTYPEAMRMODETYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nBitRate is the bitrate of the encoded AMR audio
- eAMRBandMode is the bit rate of the encoded speech. Table 4-20 shows the bit rate values.

#### **4.1.67 OMX AUDIO CONFIG BITRATETYPE**

The audio encoder's bit rate setting may be updated while the audio encoder is actively encoding, the OMX\_AUDIO\_CONFIG\_BITRATETYPE structure contains the parameters for updating the audio bit rate.

OMX\_AUDIO\_CONFIG\_BITRATETYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_BITRATETYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nEncodeBitrate;  
} OMX_AUDIO_CONFIG_BITRATETYPE;
```

#### **4.1.67.1 Parameters**

The parameters for OMX\_AUDIO\_CONFIG\_BITRATETYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nEncodeBitrate is the target bit rate for the audio encoding in units of bits per second. Encoding is set to the bitrate closest to the specified value. Use 0 to let the encoder decide on the appropriate bitrate value

#### **4.1.68 OMX AUDIO CONFIG AMRISFTYPE**

The AMR WB+ encoder's sampling frequency may be updated while the audio encoder is actively encoding.

OMX\_AUDIO\_CONFIG\_AMRISFTYPE is defined as follows.

```
typedef struct OMX_AUDIO_CONFIG_AMRISFTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_AUDIO_AMRISFINDEXTYPE eTargetAMRISFIndex;  
} OMX_AUDIO_CONFIG_AMRISFTYPE;
```

#### **4.1.68.1 Parameters**

The parameters for OMX\_AUDIO\_CONFIG\_AMRISFTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- eTargetAMRISFIndex is the target internal-sampling-frequency index for AMR-WB+ audio taking on values defined in Table 4-23. Use OMX\_AUDIO\_AMRISFIndexAuto to let the encoder decide on the appropriate ISF value. This parameter shall be ignored for formats other than AMR-WB+.

#### **4.1.69 OMX\_AUDIO\_FIXEDPOINTTYPE**

The OMX\_AUDIO\_FIXEDPOINTTYPE structure is used to set or query the current settings for the packing of PCM data within the elements specified by OMX\_AUDIO\_FIXEDPOINTTYPE.

OMX\_AUDIO\_FIXEDPOINTTYPE is defined as follows.

```
typedef struct OMX_AUDIO_FIXEDPOINTTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nValueStartBit;  
    OMX_U32 nValueBits;  
    OMX_U32 nSignExtensionBits;  
    OMX_S32 nValuePointPosition;  
} OMX_AUDIO_FIXEDPOINTTYPE;
```

#### **4.1.69.1 Parameters**

The parameters for OMX\_AUDIO\_FIXEDPOINTTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- nValueStartBit is the bit position of the lowest valid bit.
- nValueBits is the number of bits for each sample.
  - The value is contained in bits nValueStartBit + nValueBits – 1 .. nValueStartBit inclusive.
  - When extracting the bit field, sign extension will be required if, and only if, OMX\_AUDIO\_PARAM\_PCM\_MODETYPE specifies OMX\_NumbericalDataSigned.
- nSignExtensionBits is the number of additional sign bits. These shall be a copy of the sign bit. An implementation may extract the bit field with or without these bits, as the result is guaranteed to be identical.

- nValuePointPosition is the bit position of the fixed point. This may be outside the valid bits, which requires implicit bits to be added. For non-fixed point samples, this shall be nValueStartBit.

#### 4.1.69.2 Functionality

The OMX\_AUDIO\_FIXEDPOINTTYPE structure sets / gets position of the PCM sample bit field and its fixed point interpretation. Setting OMX\_AUDIO\_PARAM\_PCMMODETYPE shall implicitly set OMX\_AUDIO\_FIXEDPOINTTYPE to:

<u>Field Name</u>	<u>OMX AUDIO PARAM PCMMODETYPE</u> <u>Derived Value</u>
<u>nValueStartBit</u>	<u>0</u>
<u>nValueBits</u>	<u>nBitsPerSample</u>
<u>nSignExtensionBits</u>	<u>0</u>
<u>nValuePointPosition</u>	<u>nBitsPerSample-1</u>

The following table shows examples of common PCM formats:

	<u>Typical 16b</u>	<u>Packed 24b (in 32b)</u>
<u>OMX_AUDIO_FIXEDPOINTTYPE</u>		
<u>nValueStartBit</u>	<u>0</u>	<u>0</u>
<u>nValueBits</u>	<u>16</u>	<u>24</u>
<u>nSignExtensionBits</u>	<u>0</u>	<u>8</u>
<u>nValuePointPosition</u>	<u>15</u>	<u>23</u>
<u>OMX_AUDIO_PARAM_PCMMODETYPE</u>		
<u>nBitPerSample</u>	<u>16</u>	<u>32</u>
<u>eNumData</u>	<u>OMX_NumericalDataSigned</u>	

## 4.2 Image and Video Common

This section describes the parameter and configuration details for ports in the video and image domains. These parameter and configuration details are specified in the OMX\_IVCommon.h header.

#### 4.2.1 Uncompressed Data Formats

Both image and video ports operate on compressed and uncompressed data. The formats for uncompressed pixel data are common to both image and video. Table 4-32 lists the uncompressed formats.

Table 4-32: Uncompressed Data Formats

OMX_COLOR_FORMATTYPE	Description
OMX_COLOR_FormatUnused	Placeholder value when format is unknown, or specified using a vendor-specific means.
OMX_COLOR_FormatMonochrome	1 bit per pixel monochrome.
OMX_COLOR_FormatL2	2 bit per pixel luminance.
OMX_COLOR_FormatL4	4 bit per pixel luminance.
OMX_COLOR_FormatL8	8 bit per pixel luminance.
OMX_COLOR_FormatL16	16 bit per pixel luminance.
OMX_COLOR_FormatL24	24 bit per pixel luminance.
OMX_COLOR_FormatL32	32 bit per pixel luminance.
OMX_COLOR_Format8bitRGB332	8 bits per pixel RGB format with colors stored as Red 7:5, Green 4:2, and Blue 1:0.
<a href="#">OMX_COLOR_Format8bitBGR233</a>	<a href="#">8 bits per pixel BGR format with colors stored as Blue 7:6, Green 5:3, and Red 2:0.</a>
OMX_COLOR_Format12bitRGB444	12 bits per pixel RGB format with colors stored as Red 11:8, Green 7:4, and Blue 3:0.
<a href="#">OMX_COLOR_Format12bitBGR444</a>	<a href="#">12 bits per pixel BGR format with colors stored as Blue 11:8, Green 7:4, and Red 3:0.</a>
OMX_COLOR_Format16bitARGB4444	16 bits per pixel ARGB format with colors stored as Alpha 15:12, Red 11:8, Green 7:4, and Blue 3:0.
<a href="#">OMX_COLOR_Format16bitBGRA4444</a>	<a href="#">16 bits per pixel BGRA format with colors stored as Blue 15:12, Green 11:8, Red 7:4, and Alpha 3:0.</a>
OMX_COLOR_Format16bitARGB1555	16 bits per pixel ARGB format with colors stored as Alpha 15, Red 14:10, Green 9:5, and Blue 4:0.
<a href="#">OMX_COLOR_Format16bitBGRA5551</a>	<a href="#">16 bits per pixel BGRA format with colors stored as Blue 15:11, Green 10:6, Red 5:1, and Alpha 0.</a>

<b>OMX_COLOR_FORMATTYPE</b>	<b>Description</b>
OMX_COLOR_Format16bitRGB565	16 bits per pixel RGB format with colors stored as Red 15:11, Green 10:5, and Blue 4:0.
OMX_COLOR_Format16bitBGR565	16 bits per pixel BGR format with colors stored as Blue 15:11, Green 10:5, and Red 4:0.
OMX_COLOR_Format18bitRGB666	18 bits per pixel RGB format with colors stored as Red 17:12, Green 11:6, and Blue 5:0.
OMX_COLOR_Format18BitBGR666	18 bits per pixel BGR format with colors stored as Blue 17:12, Green 11:6, and Red 5:0.
OMX_COLOR_Format18bitARGB1665	18 bits per pixel ARGB format with colors stored as Alpha 17, Red 16:11, Green 10:5, and Blue 4:0.
<u>OMX_COLOR_Format18bitBGRA5661</u>	<u>18 bits per pixel BGRA format with colors stored as Blue 17:13, Green 12:7, Red 6:1, and Alpha 0.</u>
OMX_COLOR_Format19bitARGB1666	19 bits per pixel ARGB format with colors stored as Alpha 18, Red 17:12, Green 11:6, and Blue 5:0.
<u>OMX_COLOR_Format19bitBGRA6661</u>	<u>19 bits per pixel BGRA format with colors stored as Blue 18:13, Green 12:7, Red 6:1, and Alpha 0.</u>
OMX_COLOR_Format24bitRGB888	24 bits per pixel RGB format with colors stored as Red 23:16, Green 15:8, and Blue 7:0.
OMX_COLOR_Format24bitBGR888	24 bits per pixel BGR format with colors stored as Blue 23:16, Green 15:8, and Red 7:0.
OMX_COLOR_Format24bitARGB1887	24 bits per pixel ARGB format with colors stored as Alpha 23, Red 22:15, Green 14:7, and Blue 6:0.
<u>OMX_COLOR_Format24bitBGRA7881</u>	<u>24 bits per pixel BGRA format with colors stored as Blue 23:17, Green 16:9, Red 8:1, and Alpha 0.</u>
OMX_COLOR_Format24BitARGB6666	24 bits per pixel ARGB format with colors stored as Alpha 23:18, Red 17:12, Green 11:6, and Blue 5:0
OMX_COLOR_Format24BitABGR6666	24 bits per pixel ARGB format with colors stored as Alpha 23:18, Blue 17:12, Green 11:6, and Red 5:0

OMX_COLOR_FORMATTYPE	Description
<a href="#">OMX_COLOR_Format24BitBGRA6666</a>	24 bits per pixel BGRA format with colors stored as Blue 23:18, Green 17:12, Red 11:6, and Alpha 5:0.
<a href="#">OMX_COLOR_Format24BitRGBA6666</a>	24 bits per pixel RGBA format with colors stored as Red 23:18, Green 17:12, Blue 11:6, and Alpha 5:0.
OMX_COLOR_Format25bitARGB1888	25 bits per pixel ARGB format with colors stored as Alpha 24, Red 23:16, Green 15:8, and Blue 7:0.
<a href="#">OMX_COLOR_Format25bitBGRA8881</a>	25 bits per pixel BGRA format with colors stored as Blue 24:17, Green 16:9, Red 8:1, and Alpha 0.
OMX_COLOR_Format32bitBGRA8888	32 bits per pixel <u>BGRA</u> <u>ARGB</u> format with colors stored as <u>Blue</u> <u>Alpha</u> 31:24 <u>Green</u> <u>Red</u> 23:16, <u>Red</u> <u>Green</u> 15:8, and <u>Alpha</u> <u>Blue</u> 7:0.
OMX_COLOR_Format32bitARGB8888	24 bits per pixel <u>A</u> <u>RB</u> <u>GR</u> <u>B</u> format with colors stored as Alpha 31:24, <u>Red</u> <u>Blue</u> 23:16, Green 15:8, and <u>Blue</u> <u>Red</u> 7:0.
<a href="#">OMX_COLOR_Format32bitABGR8888</a>	32 bits per pixel ABGR format with colors stored as Alpha 31:24, Blue 23:16, Green 15:8, and Red 7:0.
OMX_COLOR_FormatYUV411Planar	YUV planar format, organized with three separate planes for each color component, namely Y, U, and V appearing in this order. U and V pixels are sub-sampled by a factor of four both horizontally and vertically.
OMX_COLOR_FormatYUV411PackedPlanar	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. This format differs from OMX_COLOR_FormatYUV411Planar in that each slice of data shall contain a plane of Y, U, and V data in this order, whereas the OMX_COLOR_FormatYUV411P1 format transfers each plane in its entirety.

<b>OMX_COLOR_FORMATTYPE</b>	<b>Description</b>
OMX_COLOR_FormatYUV420Planar	YUV planar format, organized with three separate planes for each color component, namely Y, U, and V appearing in this order. U and V pixels are sub-sampled by a factor of two both horizontally and vertically.
OMX_COLOR_FormatYUV420PackedPlanar	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. This format differs from OMX_COLOR_FormatYUV420Planar in that each slice of data shall contain a plane of Y, U, and V data in this order, whereas the OMX_COLOR_FormatYUV420Planar format transfers each plane in its entirety.
OMX_COLOR_FormatYUV420SemiPlanar	YUV planar format, organized with a first plane containing Y pixels, and a second plane containing U and V pixels interleaved with the first U value first. U and V pixels are sub-sampled by a factor of two both horizontally and vertically.
OMX_COLOR_FormatYUV420PackedSemiPlanar	YUV planar format, organized with a first plane containing Y pixels, and a second plane containing U and V pixels interleaved with the first U value first. U and V pixels are sub-sampled by a factor of two both horizontally and vertically. This format differs from OMX_COLOR_FormatYUV420SemiPlanar in that each slice of data shall contain a plane of Y, U and V data, whereas the OMX_COLOR_FormatYUV420SemiPlanar format transfers each plane in its entirety.
<u>OMX_COLOR_FormatYVU420Planar</u>	<u>YVU planar format, organized with three separate planes for each color component, namely Y, V, and U appearing in this order. V and U pixels are sub-sampled by a factor of two both horizontally and vertically.</u>

<b>OMX_COLOR_FORMATTYPE</b>	<b>Description</b>
<a href="#"><u>OMX_COLOR_FormatYVU420PackedPlanar</u></a>	<p><u>YVU planar format, organized with three separate planes for each color component, namely Y, V, and U.</u> <u>V and U pixels are sub-sampled by a factor of two both horizontally and vertically.</u></p> <p>This format differs from <u>OMX_COLOR_FormatYVU420Planar</u> in that each slice of data shall contain a plane of Y, <u>V</u>, and <u>U</u> data in this order, whereas the <u>OMX_COLOR_FormatYVU420Planar</u> format transfers each plane in its entirety.</p>
<a href="#"><u>OMX_COLOR_FormatYVU420SemiPlanar</u></a>	<p><u>YVU planar format, organized with a first plane containing Y pixels, and a second plane containing V and U pixels interleaved with the first V value first.</u> <u>V and U pixels are sub-sampled by a factor of two both horizontally and vertically.</u></p>
<a href="#"><u>OMX_COLOR_FormatYVU420PackedSemiPlanar</u></a>	<p><u>YVU planar format, organized with a first plane containing Y pixels, and a second plane containing V and U pixels interleaved with the first V value first.</u> <u>V and U pixels are sub-sampled by a factor of two both horizontally and vertically.</u></p> <p>This format differs from <u>OMX_COLOR_FormatYVU420SemiPlanar</u> in that each slice of data shall contain a plane of Y, V and U data, whereas the <u>OMX_COLOR_FormatYVU420SemiPlanar</u> format transfers each plane in its entirety.</p>
<a href="#"><u>OMX_COLOR_FormatYUV422Planar</u></a>	<p><u>YUV planar format, organized with three separate planes for each color component, namely Y, U, and V appearing in this order.</u> <u>U and V pixels are sub-sampled by a factor of two horizontally.</u></p>

<b>OMX_COLOR_FORMATTYPE</b>	<b>Description</b>
OMX_COLOR_FormatYUV422PackedPlanar	<p>YUV planar format, organized with three separate planes for each color component, namely Y, U, and V. <u>U and V pixels are sub-sampled by a factor of two horizontally.</u></p> <p>This format differs from OMX_COLOR_FormatYUV422Planar in that each slice of data shall contain a plane of Y, U, and V data in this order, whereas the OMX_COLOR_FormatYUV422Planar format transfers each plane in its entirety.</p>
OMX_COLOR_FormatYUV422SemiPlanar	<p>YUV planar format, organized with a first plane containing Y pixels and a second plane containing U and V pixels interleaved with the first U value first. <u>U and V pixels are sub-sampled by a factor of two horizontally.</u></p>
OMX_COLOR_FormatYUV422PackedSemiPlanar	<p>YUV planar format, organized with a first plane containing Y pixels, and a second plane containing U and V pixels interleaved with the first U value first. U and V pixels are sub-sampled by a factor of two horizontally.</p> <p>This format differs from OMX_COLOR_FormatYUV422SemiPlanar in that each slice of data shall contain a plane of Y, U and V data, whereas the OMX_COLOR_FormatYUV422SemiPlanar format transfers each plane in its entirety.</p>
<u>OMX_COLOR_FormatYVU422Planar</u>	<p><u>YVU planar format, organized with three separate planes for each color component, namely Y, V, and U appearing in this order. V and U pixels are sub-sampled by a factor of two horizontally.</u></p>

<b>OMX_COLOR_FORMATTYPE</b>	<b>Description</b>
<a href="#"><u>OMX_COLOR_FormatYVU422PackedPlanar</u></a>	<p>YVU planar format, organized with three separate planes for each color component, namely Y, V, and U appearing in this order. V and U pixels are sub-sampled by a factor of two horizontally.</p> <p>This format differs from <a href="#"><u>OMX_COLOR_FormatYVU422Planar</u></a> in that each slice of data shall contain a plane of Y, V, and U data in this order, whereas the <a href="#"><u>OMX_COLOR_FormatYVU422Planar</u></a> format transfers each plane in its entirety.</p>
<a href="#"><u>OMX_COLOR_FormatYVU422SemiPlanar</u></a>	<p>YVU planar format, organized with a first plane containing Y pixels and a second plane containing V and U pixels interleaved with the first V value first. V and U pixels are sub-sampled by a factor of two horizontally.</p>
<a href="#"><u>OMX_COLOR_FormatYVU422PackedSemiPlanar</u></a>	<p>YVU planar format, organized with a first plane containing Y pixels, and a second plane containing V and U pixels interleaved with the first V value first. V and U pixels are sub-sampled by a factor of two horizontally.</p> <p>This format differs from <a href="#"><u>OMX_COLOR_FormatYVU422SemiPlanar</u></a> in that each slice of data shall contain a plane of Y, V and U data, whereas the <a href="#"><u>OMX_COLOR_FormatYVU422SemiPlanar</u></a> format transfers each plane in its entirety.</p>
OMX_COLOR_FormatYCbYCr	16 bits per pixel YUV interleaved format organized as YUYV (i.e., YCbYCr).
OMX_COLOR_FormatYCrYCb	16 bits per pixel YUV interleaved format organized as YVYU (i.e., YCrYCb).
OMX_COLOR_FormatCbYCrY	16 bits per pixel YUV interleaved format organized as UYVY (i.e., CbYCrY).

<b>OMX_COLOR_FORMATTYPE</b>	<b>Description</b>
OMX_COLOR_FormatCrYCbY	16 bits per pixel YUV interleaved format organized as VYUY (i.e., CrYCbY).
OMX_COLOR_FormatYUV444Interleaved	12 bits per pixel YUV format with colors stores as Y 11:8, U 7:4, and V 3:0.
OMX_COLOR_FormatRawBayer8bit	SMIA 8-bit raw Bayer pattern camera format.
OMX_COLOR_FormatRawBayer10bit	SMIA 10-bit raw Bayer pattern camera format.
OMX_COLOR_FormatRawBayer8bitcompressed	SMIA compressed 8-bit camera output format.

#### **4.2.2 Minimum Buffer Payload Size for Uncompressed Data**

Uncompressed image and video data have a minimum buffer payload size. The minimum buffer payload size is determined by the nSliceHeight and nStride fields of the port definition structure. nStride indicates the width of a span in bytes; when negative, it indicates the data is bottom-up instead of the top-down). nSliceHeight indicates the number of spans in a slice.

The minimum buffer payload size can be easily calculated as the absolute value of (nSliceHeight \* nStride ).

#### **4.2.3 Buffer Payload Requirements for Uncompressed Data**

Each image or video port on a component shall meet several requirements for buffer payloads of uncompressed image and video data. These requirements are in place to enable components from different vendors to inter-operate together correctly, and are collectively referred to as *inter-op*.

The requirements are as follows:

- Each non-empty buffer payload shall contain at least one full slice, unless it contains the end of the image (which may not be aligned to an integer multiple of slice height). For example, if the image height is 100 and the slice height is 16, the last slice of the image will contain only four spans.
- Each non-empty buffer payload shall contain an integer multiple of slice height.
- When the uncompressed image data format is planar, data from two different planes cannot reside in the same buffer payload. This means that a component shall pass a full plane in its entirety in one or more buffers, followed by another plane starting in a different buffer.

An exception to the above requirement exists for the packed planar uncompressed formats, OMX\_COLOR\_FormatYUV420PackedPlanar,  
OMX\_COLOR\_FormatYUV420PackedSemiPlanar,

OMX\_COLOR\_FormatYVU420PackedPlanar,  
OMX\_COLOR\_FormatYVU420PackedSemiPlanar,  
OMX\_COLOR\_FormatYUV411PackedPlanar,  
OMX\_COLOR\_FormatYUV422PackedPlanar, and  
OMX\_COLOR\_FormatYUV422PackedSemiPlanar,  
OMX\_COLOR\_FormatYVU422PackedPlanar, and  
OMX\_COLOR\_FormatYVU422PackedSemiPlanar.

For each of these uncompressed formats, each buffer payload contains a slice of the Y, U, and V planes. The slices are always ordered Y, U, and V or Y, V and U - depending on their color format definition. The nSliceHeight refers to the slice height of the Y plane. The slice height of the U and V planes are derived from the slice height for the Y plane based upon the format. For example, for OMX\_COLOR\_FormatYUV420PackedPlanar with an nSliceHeight of 16, a buffer payload shall contain 16 spans of Y followed by 8 spans of U (half the stride) and 8 spans of V (half the stride). This enables ports that process planar data in slices to operate on all three planes simultaneously, instead of forcing the ports to buffer the entire image before processing can begin.

#### 4.2.4 Parameter and Configuration Indexes

The header OMX\_Index.h contains the enumeration OMX\_INDEXTYPE, which contains all of the standard index values used with the functions OMX\_GetParameter, OMX\_SetParameter, OMX\_GetConfig, and OMX\_SetConfig. Table 4-33 describes the index values that relate to video.

**Table 4-33: Index Values for Video**

Index	Description
OMX_IndexParamCommonDeblocking	Used with OMX_GetParameter and OMX_SetParameter to access OMX_PARAM_DEBLOCKINGTYPE. Deblocking reduces the appearance of block-like artifacts that appear in compressed images or video streams.
OMX_IndexParamCommonSensorMode	Used with OMX_GetParameter and OMX_SetParameter to access OMX_PARAM_SENSORMODETYPE. The mode of the sensor controls the resolution ( <u>via OMX_FRAMESIZETYPE</u> ) and frame rate of data captured by a camera.
OMX_IndexParamCommonInterleave	Used with OMX_GetParameter and OMX_SetParameter to access OMX_PARAM_INTERLEAVETYPE. This feature is used to interleave plane or input port data.

Index	Description
OMX_IndexConfigCommonColorFormatConversion	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_COLORCONVERSIONTYPE. Color conversion programs the coefficients used when converting pixel data from RGB to YUV and visa-versa.
OMX_IndexConfigCommonScale	Used with OMX_GetConfig and OMX_SetConfig to access the OMX_CONFIG_SCALEFACTORTYPE. Scaling stretches or shrinks a rectangular region of pixels.
OMX_IndexConfigCommonImageFilter	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_IMAGEFILERTYPE. Image filtering applies digital effects to a video or image stream.
OMX_IndexConfigCommonColorEnhancement	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_COLORENHANCEMENTTYPE. Color enhancement replaces U and V values of a YUV image with specified constant values to apply a color effect to an image or video stream.
OMX_IndexConfigCommonColorKey	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_COLORKEYTYPE. Color keying performs per-pixel selection between two sources with mixing image or video data.
OMX_IndexConfigCommonColorBlend	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_COLORBLENDTYPE. Color blending performs arithmetic operations between two sources.
OMX_IndexConfigCommonFrameStabilisation	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_FRAMESTABTYPE.
OMX_IndexConfigCommonRotate	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_ROTATIONTYPE. Rotation rotates video or image frames clockwise by a specified angle.

Index	Description
OMX_IndexConfigCommonMirror	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_MIRRORTYPE. Mirroring reflects video or image frames along the horizontal and vertical axes.
OMX_IndexConfigCommonOutputPosition	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_POINTTYPE. The output position indicates the location of a video or image stream relative to another image or video stream. The output position is also used to indicate the location of a video or image stream relative to an output device such as an LCD display.
OMX_IndexConfigCommonInputCrop	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_RECTTYPE. Crops the image or video stream to the specified rectangle.
OMX_IndexConfigCommonOutputCrop	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_RECTTYPE. Crops the image or video stream to the specified rectangle.
OMX_IndexConfigCommonDigitalZoom	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_SCALEFACTORTYPE. Digital zoom implements a camera zoom feature digitally.
OMX_IndexConfigCommonOpticalZoom	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_SCALEFACTORTYPE. Optical zoom “zooms” an image in or out using a lens on a camera.
OMX_IndexConfigCommonWhiteBalance	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_WHITEBALCONTROLTYPE. White balance performs color correction so that a white object appears truly white and not a tint of the color of the light source.

Index	Description
OMX_IndexConfigCommonExposure	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_EXPOSURECONTROLYPE. Exposure controls the image sensor exposure when capturing images or streaming video.
OMX_IndexConfigCommonContrast	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_CONTRASTTYPE. Contrast controls the relative difference between pixels in video or image data.
OMX_IndexConfigCommonBrightness	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_BRIGHTNESSTYPE. Brightness controls the luminosity of the pixels in video or image data.
OMX_IndexConfigCommonBacklight	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_BACKLIGHTTYPE. Backlight controls the strength of the backlight, and the time that the backlight is turned on.
OMX_IndexConfigCommonGamma	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_GAMMATYPE. Gamma corrects for the non-linear intensity of pixels on a display relative to the digital value of the pixel for video or image data.
OMX_IndexConfigCommonSaturation	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_SATURATIONTYPE. Saturation controls the hue intensity of video or image data.
OMX_IndexConfigCommonLightness	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_LIGHTNESSTYPE. Lightness controls the non-linear response to the brightness of pixels in video or image data.

Index	Description
OMX_IndexConfigCommonExclusionRect	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_RECTTYPE. This feature enables a component to exclude a specific region from rendering to save on processing, resulting in higher performance and lower power consumption. This configuration is often used in video conferencing where a section of the decoded input stream is covered by a preview of the viewer's image.
OMX_IndexConfigCommonPlaneBlend	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_PLANEBLENDTYPE. This feature controls the blending of multiple input sources or ports into a single destination.
OMX_IndexConfigCommonTransitionEffect	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_TRANSITIONEFFECTTYPE.
OMX_IndexConfigCommonDithering	Used with OMX_GetConfig and OMX_SetConfig to access OMX_CONFIG_DITHERTYPE. Dithering is used when performing color space conversion from a color format that has a higher precision to a color format with a lower precision.
OMX_IndexConfigCommonExposureValue	OMX_CONFIG_EXPOSUREVALUETYPE Query or config the exposure value of the camera.
OMX_IndexConfigCommonOutputSize	OMX_FRAMESIZETYPE Query or config the frame size of an output video sink region.
OMX_IndexParamCommonExtraQuantData	OMX_OTHER_EXTRADATATYPE Used to enable or query the generation of extra payload information consisting of quantization information.
OMX_IndexConfigCaptureMode	OMX_CONFIG_CAPTUREMODETYPE Query or config the capture mode of a camera.

Index	Description
<a href="#">OMX_IndexAutoPauseAfterCapture</a>	OMX_CONFIG_BOOLEANTYPE Query or config the auto pause mechanism after capturing is complete for a camera.
<a href="#">OMX_IndexConfigCapturing</a>	OMX_CONFIG_BOOLEANTYPE Query a component if it is capturing data.
<a href="#">OMX_IndexConfigCommonFocusRegion</a>	<del>OMX_CONFIG_FOCUSREGIONTYPE</del> <del>Query or config the focus regions of interest.</del>
<a href="#">OMX_IndexConfigCommonFocusStatus</a>	<del>OMX_PARAM_FOCUSSTATUSTYPE</del> <del>Query the focus status of the individual focus regions.</del>
<a href="#">OMX_IndexConfigSharpness</a>	OMX_SHARPNESSTYPE <u>Increasing negative values indicate increased blurriness while increasing positive values indicate increased sharpness.</u>
<a href="#">OMX_IndexConfigCommonExtDigitalZoom</a>	OMX_CONFIG_ZOOMFACTORTYPE. <u>Digital zoom implements a camera zoom feature digitally.</u>
<a href="#">OMX_IndexConfigCommonExtOpticalZoom</a>	OMX_CONFIG_ZOOMFACTORTYPE. <u>Optical zoom implements a camera zoom using optical lens. feature digitally.</u>
<a href="#">OMX_IndexConfigCommonCenterFieldOfView</a>	OMX_CONFIG_POINTTYPE  <u>Configures the field of view that is associated with Digital zoom.</u>  <u>Allows setting and querying the center of field of view in case of digital zoom, relatively to the center of the observed scene.</u> <u>By default, the center of the field of view is the center of the observed scene.</u>  <u>See OMX_CONFIG_ZOOMFACTORTYPE for more details.</u>
<a href="#">OMX_IndexConfigImageExposureLock</a>	OMX_IMAGE_CONFIG_LOCKTYPE <u>Allows locking the exposure.</u>
<a href="#">OMX_IndexConfigImageWhiteBalanceLock</a>	OMX_IMAGE_CONFIG_LOCKTYPE <u>Allows locking the white balance.</u>
<a href="#">OMX_IndexConfigImageFocusLock</a>	OMX_IMAGE_CONFIG_LOCKTYPE <u>Allows locking the focus.</u>

Index	Description
<a href="#">OMX_IndexConfigCommonFocusRange</a>	<a href="#">OMX_CONFIG_FOCUSRANGETYPE</a>  Allows setting the focus range.
<a href="#">OMX_IndexConfigImageFlashStatus</a>  <u>This is a read only config.</u>	<a href="#">OMX_IMAGE_CONFIG_FLASHSTATUSTYPE</a>  Provides status of the flash (read only).
<a href="#">OMX_IndexConfigCommonExtCaptureMode</a>	<a href="#">OMX_CONFIG_EXTCAPTUREMODETYPE</a>  Configures extended capture mode settings.
<a href="#">OMX_IndexConfigCommonNDFilterControl</a>	<a href="#">OMX_CONFIG_NDFILTERCONTROLTYP</a> E  Allows controlling the ND Filter functionality.
<a href="#">OMX_IndexConfigCommonAFAssistantLight</a>	<a href="#">OMX_CONFIG_AFASSISTANTLIGHTTYPE</a>  Allows controlling a light assistant during camera focusing.
<a href="#">OMX_IndexParamInterlaceFormat</a>	<a href="#">OMX_INTERLACEFORMATTYPE</a>  Used to query the supported interlace formats.
<a href="#">OMX_IndexConfigDeInterlace</a>	<a href="#">OMX_DEINTERLACETYPE</a>  Used to enable disable deinterlacing support.
<a href="#">OMX_IndexConfigStreamInterlaceFormats</a>	<a href="#">OMX_STREAMINTERLACEFORMATTYPE</a>  Used to query if the stream contains interlace or progressive content.
<a href="#">OMX_IndexConfigCommonPortCapturing</a>	<a href="#">OMX_CONFIG_PORTBOOLEANTYPE</a> . Query a component if it is capturing data.
<a href="#">OMX_IndexConfigCommonFocusRegionStatus</a>	<a href="#">OMX_CONFIG_FOCUSREGIONSTATUSTYPE</a>  Allows retrieving the status of focusing areas.
<a href="#">OMX_IndexConfigCommonFocusRegionControl</a>	<a href="#">OMX_CONFIG_FOCUSREGIONCONTROLTYPE</a>  Allows setting/getting the control of focusing areas.

Index	Description
<u><a href="#">OMX_IndexParamInterlaceFormat</a></u>	<u><a href="#">OMX_INTERLACEFORMATTYPE</a></u> Used to query the supported interlace formats.
<u><a href="#">OMX_IndexConfigDeInterlace</a></u>	<u><a href="#">OMX_DEINTERLACETYPE</a></u> Used to enable/disable deinterlacing support.
<u><a href="#">OMX_IndexConfigStreamInterlaceFormats</a></u>	<u><a href="#">OMX_STREAMINTERLACEFORMATTYPE</a></u> Used to query if the stream contains interlace or progressive content.

#### 4.2.5 **OMX\_PARAM\_DEBLOCKINGTYPE**

De-blocking is used to reduce the appearance of block-like artifacts that appear in compressed images or video streams.

`OMX_PARAM_DEBLOCKINGTYPE` is defined as follows.

```
typedef struct OMX_PARAM_DEBLOCKINGTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bDeblocking;
} OMX_PARAM_DEBLOCKINGTYPE;
```

##### 4.2.5.1 **Parameters**

The parameters for `OMX_PARAM_DEBLOCKINGTYPE` are defined as follows.

- `nPortIndex` is the read-only value containing the index of the port represents the port that this structure applies to.
- `bDeblocking` is a Boolean value that enables or disables de-blocking.

#### 4.2.6 **OMX\_PARAM\_INTERLEAVETYPE**

Interleaving is used to interleave or de-interleave pixel data between multiple ports.

When interleaving, a component uses pixel data from multiple input ports to merge into a single output port. When de-interleaving, a component uses pixel data from a single input port, splitting the color channels into separate output ports.

For example, an input port receiving 16-bit RGB can de-interleave R, G, and B color channels to three separate output ports, where the output ports are formatted as monochrome.

Similarly, a component could interleave three luminance ports containing Y, U, and V data into a single output port formatted as YUV420.

The OMX\_PARAM\_INTERLEAVETYPE structure interleaves pixel data.

OMX\_PARAM\_INTERLEAVETYPE is defined as follows.

```
typedef struct OMX_PARAM_INTERLEAVETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnable;
    OMX_U32 nInterleavePortIndex;
} OMX_PARAM_INTERLEAVETYPE;
```

#### 4.2.6.1 Parameters

The parameters for OMX\_PARAM\_INTERLEAVETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- bEnable is a Boolean value that enables interleaving.
- nInterleavePortIndex indicates the port to interleave or de-interleave with. When nPortIndex is an input port, nInterleavePortIndex contains the output port to interleave with. When nPortIndex is an output port, nInterleavePortIndex contains the input port to de-interleave with.

#### 4.2.7 OMX\_PARAM\_SENSORMODETYPE

The sensor mode is used to specify the frame rate and resolution that an image sensor or camera uses to capture image or video. The sensor mode is distinctly separate from the port on a video source, which may modify the resolution of the data produced by the image sensor.

OMX\_PARAM\_SENSORMODETYPE is defined as follows.

```
typedef struct OMX_PARAM_SENSORMODETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nFrameRate;
    OMX_BOOL bOneShot;
    OMX_FRAMESETTYPE sFrameSize;
} OMX_PARAM_SENSORMODETYPE;
```

#### 4.2.7.1 Parameters

The parameters for OMX\_PARAM\_SENSORMODETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.

- `nFrameRate` is the frame rate is in frames per second. This value is represented in Q16 format. The value 0x0 is used to indicate the frame rate is unknown, variable, or is not needed.
- `bOneShot` is a Boolean value that enables or disables one shot mode.
- `sFrameSize` is the resolution of the image sensor mode provided in the form of `OMX_FRAMESIZETYPE`.

#### **4.2.8 OMX\_FRAMESIZETYPE**

Frame size is a generic structure used to indicate the size of a frame. This structure is referred to by the `OMX_PARAM_SENSORMODETYPE` structure.

`OMX_FRAMESIZETYPE` is defined as follows.

```
typedef struct OMX_FRAMESIZETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nWidth;
    OMX_U32 nHeight;
} OMX_FRAMESIZETYPE;
```

##### **4.2.8.1 Parameters**

The parameters for `OMX_FRAMESIZETYPE` are defined as follows.

- `nPortIndex` represents the port that this structure applies to.
- `nWidth` is the width of the rectangle in pixels.
- `nHeight` is the height of the rectangle in pixels.

#### **4.2.8.2.9 OMX\_CONFIG\_COLORCONVERSIONTYPE**

Color conversion is used to specify the coefficients when converting image or video pixel data from YUV to RGB and visa-versa.

Converting from RGB to YUV format uses the following standard formulae:

$$Y = 0.299R + 0.587G + 0.114B$$

$$U = -0.147R - 0.289G + 0.436B$$

$$V = 0.615R - 0.515G - 0.100B$$

Converting from YUV to RGB format uses the following standard formulae:

$$R = Y + 1.140V$$

$$G = Y - 0.395U - 0.581V$$

$$B = Y + 2.032U$$

The color matrix and color offset specified in the color conversion allow for the coefficients used when converting from RGB to YUV and visa-versa to be programmed explicitly.

`OMX_CONFIG_COLORCONVERSIONTYPE` is defined as follows.

```
typedef struct OMX_CONFIG_COLORCONVERSIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 xColorMatrix[3][3];
    OMX_S32 xColorOffset[4];
} OMX_CONFIG_COLORCONVERSIONTYPE;
```

#### 4.2.8.14.2.9.1 Parameters

The parameters for `OMX_CONFIG_COLORCONVERSIONTYPE` are defined as follows.

- `nPortIndex` ~~represents the port that this structure applies to is the read-only field indicating the index of the port.~~
- `xColorMatrix[3][3]` is the color conversion matrix when converting from RGB to YUV in Q16 format with the following standard formulae:

$$\begin{aligned}Y &= Y_r * R + Y_g * G + Y_b * B \\U &= U_r * R - U_g * G + U_b * B \\V &= V_r * R - V_g * G - V_b * B\end{aligned}$$

Each constant is represented in the 3x3 matrix as:

$$\begin{matrix}Y_r & Y_g & Y_b \\U_r & U_g & U_b \\V_r & V_g & V_b\end{matrix}$$

Y constants are in the first row, followed by U and V constants in subsequent rows. All constants multiplied against red color values are in the first column followed by green and blue color constants, as follows

```
xColorMatrix[1][1] = Yr
xColorMatrix[3][3] = Vb,
xColorMatrix[1][3] = Yb
```

- `xColorOffset[4]` is the color conversion vector when converting from YUV to RGB in Q16 format. The standard formulae are as follows:

$$\begin{aligned}R &= Y + C1 * U \\G &= Y - C2 * U - C3 * V \\B &= Y - C4 * V\end{aligned}$$

Each constant is represented in the array:

$$C1 \ C2 \ C3 \ C4$$

#### **4.2.94.2.10 OMX\_CONFIG\_SCALEFACTORTYPE**

Scaling is used to stretch or shrink video or image data on the specified input or output port.

OMX\_CONFIG\_SCALEFACTORTYPE is defined as follows.

```
typedef struct OMX_CONFIG_SCALEFACTORTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 xWidth;
    OMX_S32 xHeight;
} OMX_CONFIG_SCALEFACTORTYPE;
```

##### **4.2.9.14.2.10.1 Parameters**

The parameters for OMX\_CONFIG\_SCALEFACTORTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- xWidth is the scaling in the horizontal direction in Q16 format (i.e., signed 15.16 fixed pointed format). For example, a scaling factor of 0x10000 would not change the width, but a scaling factor of 0x8000 would scale the width by 50%.
- xHeight is the scaling in the vertical direction in Q16 format (i.e., signed 15.16 fixed pointed format). For example, a scaling factor of 0x10000 would not change the height, but a scaling factor of 0x20000 would scale the height by 200%.

#### **4.2.104.2.11 OMX\_CONFIG\_IMAGEFILTERTYPE**

Image filtering is used to apply digital effects to video or image data on the specified port.

OMX\_CONFIG\_IMAGEFILTERTYPE is defined as follows.

```
typedef struct OMX_CONFIG_IMAGEFILTERTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_IMAGEFILTERTYPE eImageFilter;
} OMX_CONFIG_IMAGEFILTERTYPE;
```

##### **4.2.10.14.2.11.1 Parameters**

The parameters for OMX\_CONFIG\_IMAGEFILTERTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- eImageFilter is the enumerated valued indicating the image filter used. Table 4-34 details the values that can be selected for the image filter.

**Table 4-34: Image Filter Values**

<b>OMX_IMAGEFILTERTYPE Enumerated Value</b>	<b>Description</b>
OMX_ImageFilterNone	Used to disable image filtering.
OMX_ImageFilterNoise	Filters data to remove noise from the image.
OMX_ImageFilterEmboss	Filters data to give an embossed appearance (stamped from the rear for a raised effect along edges).
OMX_ImageFilterNegative	Filters data to negate colors.
OMX_ImageFilterSketch	Filters data to give the appearance of having been sketched by an artist.
OMX_ImageFilterOilPaint	Filters data to appear as if it were hand painted using a brush with oil paints.
OMX_ImageFilterHatch	Filters data to appear as if it were printed on a material with a grain.
OMX_ImageFilterPen	Filters data to appear as if it were drawn with a pen.
OMX_ImageFilterAntialias	Filters data to anti-alias pixels so as to sharpen edges in the image or video stream.
OMX_ImageFilterDeRing	Filters data to remove erroneous artifacts introduced by inherent limitations of the numerical processing of digital image data.
OMX_ImageFilterSolarize	Filters data to create a solarization effect.
<u>OMX_ImageFilterPastel</u>	<u>Filters data to provide a pastel appearance.</u>
<u>OMX_ImageFilterMosaic</u>	<u>Filters data to provide a mosaic appearance.</u>
<u>OMX_ImageFilterPosterize</u>	<u>Filters data to replace gradual transitions of tone with abrupt changes in grade and shading.</u>
<u>OMX_ImageFilterWhiteboard</u>	<u>Filters data to emphasize symbols written on a whiteboard.</u>
<u>OMX_ImageFilterBlackboard</u>	<u>Filters data to emphasize symbols written on a blackboard.</u>
<u>OMX_ImageFilterSepia</u>	<u>Filters data to provide a sepia appearance.</u>
<u>OMX_ImageFilterGrayscale</u>	<u>Filters data to provide a gray scale appearance – Black and White.</u>
<u>OMX_ImageFilterNatural</u>	<u>Filters data to provide a natural appearance.</u>
<u>OMX_ImageFilterVivid</u>	<u>Filters data to provide a vivid appearance.</u>
<u>OMX_ImageFilterWaterColor</u>	<u>Filters data to provide a water color appearance.</u>
<u>OMX_ImageFilterFilm</u>	<u>Filters data to provide a film appearance.</u>

#### **4.2.114.2.12      OMX\_CONFIG\_COLORENHANCEMENTTYPE**

Color enhancement is applied to image or video data in YUV formats, where the U and V color components of each pixel are replaced with the specified values. Replacement occurs for each pixel and every frame. This enables a component to add specified color

hues to the data. For example, this configuration can be used to convert color image or video data to sepia tone.

OMX\_CONFIG\_COLORENHANCEMENTTYPE is defined as follows.

```
typedef struct OMX_CONFIG_COLORENHANCEMENTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bColorEnhancement;
    OMX_U8 nCustomizedU;
    OMX_U8 nCustomizedV;
} OMX_CONFIG_COLORENHANCEMENTTYPE;
```

#### **4.2.11.14.2.12.1 Parameters**

The parameters for OMX\_CONFIG\_COLORENHANCEMENTTYPE are defined as follows.

- nPortIndex **is the read-only value containing the index of the portrepresents the port that this structure applies to.**
- bColorEnhancement is the Boolean value that enables or disables color enhancement.
- nCustomizedU is a value for replacing the U color component of each pixel. The range of values is 0-255. Practical values are in the range of 16-240.
- nCustomizedV is the value for replacing the V color component of each pixel. The range of values is 0-255. Practical values are in the range of 16-240.

#### **4.2.12.4.2.13 OMX\_CONFIG\_COLORKEYTYPE**

Color keying is used to perform per-pixel selection between two sources when mixing image or video data.

OMX\_CONFIG\_COLORKEYTYPE is defined as follows.

```
typedef struct OMX_CONFIG_COLORKEYTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nARGBColor;
    OMX_U32 nARGBMask;
} OMX_CONFIG_COLORKEYTYPE;
```

#### **4.2.12.14.2.13.1 Parameters**

The parameters for OMX\_CONFIG\_COLORKEYTYPE are defined as follows.

- nPortIndex **is the read-only value containing the index of the portrepresents the port that this structure applies to.**
- nARGBColor indicates a 32-bit color used for keying, where bits 0-7 are blue, bits 15-8 are green, bits 24-16 are red, and bits 31-24 are for alpha. The 32-bit

ARGB color is converted to the RGB color format of the port before performing keying operations.

- nARGBMask indicates a 32-bit logical AND mask, which is converted to the RGB color format of the port before performing keying operations.

#### **4.2.134.2.14    OMX\_CONFIG\_COLORBLENDTYPE**

Color blending is used to perform arithmetic operations between two sources when mixing image or video data. If more than one input port (representing a plane) on a component is using this config, it should be used in conjunction with OMX\_CONFIG\_PLANEBLENDTYPE to specify the Z-order of the different ports via the nDepth field.

OMX\_CONFIG\_COLORBLENDTYPE is defined as follows.

```
typedef struct OMX_CONFIG_COLORBLENDTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nRGBAlphaConstant;  
    OMX_COLORBLENDTYPE eColorBlend;  
} OMX_CONFIG_COLORBLENDTYPE;
```

#### **4.2.13.14.2.14.1    Parameters**

The parameters for OMX\_CONFIG\_COLORBLENDTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nRGBAlphaConstant is the 32-bit per color channel constant alpha value for blending when the eColorBlend is set to OMX\_ColorBlendAlphaConstant on an input port. If defined on an output port, the nRGBAlphaConstant value is written as the per pixel alpha value in the composed image (if the output format supports per pixel alpha). If eColorBlend is OMX\_ColorBlendAlphaPerPixel is defined, the nRGBAlphaConstant value is ignored and the alpha coefficients for the output buffer are taken from the corresponding alpha values of the lowest nDepth (=highest value) input plane.  
A value of 0 means fully transparent and a value of 1 (0xFFFFFFFF) means opaque.
- eColorBlend is the enumerated value indicating the color blend operation used. eColorBlend is only valid when set on ports representing the image source input (highest nDepth (=lowest value) plane) or on the composed plane. If set on an output port, assuming the output format supports per pixel alpha, the nRGBAlphaConstant value is taken (with eColorBlend = OMX\_ColorBlendAlphaConstant) or the alpha value of the lowest nDepth plane is taken (eColorBlend =

`OMX_ColorBlendAlphaPerPixel`), as per pixel alpha value in the composed image. Note in the latter case a) if the input (alpha) format does not equal the composed image (alpha) format, the implicit color space conversion takes care of re-calculating the alpha value, and b) if the input format does not have an alpha value, the per pixel alpha value of the composed plane is set to non-transparent. Table 4-35 details the values that can be selected for color blending.

**Table 4-35: Color Blending Values**

OMX_COLORBLENDTYPE Enumerated Value	Description
<code>OMX_ColorBlendNone</code>	Disables color blending.
<code>OMX_ColorBlendAlphaConstant</code>	Blends source and destination using the function $(\text{alpha\_constant} * \text{source}) + ((1 - \text{alpha\_constant}) * \text{destination})$ , where the alpha constant is specified for the entire operation.
<code>OMX_ColorBlendAlphaPerPixel</code>	Blends source and destination using the function $(\text{alpha} * \text{source}) + ((1 - \text{alpha}) * \text{destination})$ , where the alpha value is per pixel.
<code>OMX_ColorBlendAlternate</code>	Alternates between selecting source and destination pixels (i.e., checkerboard of source and destination pixels).
<code>OMX_ColorBlendAnd</code>	Combines source and destination pixels using the function $(\text{source} \& \text{destination})$ .
<code>OMX_ColorBlendOr</code>	Combines source and destination pixels using the function $(\text{source}   \text{destination})$ .
<code>OMX_ColorBlendInvert</code>	Combines source and destination pixels using the function $\sim(\text{source})$ .

#### 4.2.14 OMX\_FRAMEFORMATTYPE

Frame size is a generic structure used to indicate the size of a frame. This structure is referred to by the `OMX_PARAM_SENSORMODETYPE` structure.

`OMX_FRAMEFORMATTYPE` is defined as follows.

```
typedef struct OMX_FRAMEFORMATTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nWidth;
    OMX_U32 nHeight;
} OMX_FRAMEFORMATTYPE;
```

#### **4.2.14.1 Parameters**

The parameters for OMX\_FRAMESIZETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nWidth is the width of the rectangle in pixels.
- nHeight is the height of the rectangle in pixels.

#### **4.2.15 OMX\_CONFIG\_FRAMESTABTYPE**

Frame stabilization reduces motion blur during image capture or video recording. Frame stabilization is most often associated with camera sensor source components, a camera sensor filter, or a digital signal processor (DSP).

The frame stabilization feature compensates for the extremely unsteady nature of cameras on handheld devices such as a cell phone or personal digital assistant (PDA).

OMX\_CONFIG\_FRAMESTABTYPE is defined as follows.

```
typedef struct OMX_CONFIG_FRAMESTABTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BOOL bStab;  
} OMX_CONFIG_FRAMESTABTYPE;
```

#### **4.2.15.1 Parameters**

The parameters for OMX\_CONFIG\_FRAMESTABTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- bStab is the Boolean value that enables or disables frame stabilization.

#### **4.2.154.2.16 OMX\_CONFIG\_ROTATIONTYPE**

Rotation is applied to image or video data on a specified port. Components may support rotation only on right angles such as 0°, 90°, 180°, and 270°, although components may support arbitrary rotation angles. Values are interpreted as clockwise.

OMX\_CONFIG\_ROTATIONTYPE is defined as follows.

```
typedef struct OMX_CONFIG_ROTATIONTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_S32 nRotation;  
} OMX_CONFIG_ROTATIONTYPE;
```

#### **4.2.15.14.2.16.1 Parameters**

The parameters for OMX\_CONFIG\_ROTATIONTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nRotation is an integer value that represents the angle of rotation. Some components may only support rotation on right angles such as 0°, 90°, 180°, and 270°. Positive value of nRotation indicates a clockwise rotation is clockwise.

#### **4.2.164.2.17 OMX\_CONFIG\_MIRRORTYPE**

Mirroring is applied to pixel or image data on a specified port. The data can be mirrored in the horizontal direction, vertical direction, or both horizontal and vertical directions.

OMX\_CONFIG\_MIRRORTYPE is defined as follows.

```
typedef struct OMX_CONFIG_MIRRORTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_MIRRORTYPE eMirror;  
} OMX_CONFIG_MIRRORTYPE;
```

#### **4.2.16.14.2.17.1 Parameters**

The parameters for OMX\_CONFIG\_MIRRORTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- eMirror contains the enumerated values indicating the mirroring applied to image or video data. OMX\_MirrorNone is used to disable mirroring or have no mirroring. Table 4-36 identifies the mirroring values.

**Table 4-36: Mirror Type Values**

<b>OMX_MIRRORTYPE Enumerated Value</b>	<b>Description</b>
OMX_MirrorNone	Disables mirroring (i.e., no mirroring).
OMX_MirrorHorizontal	Mirrors pixels in the horizontal direction. Hence, pixel at 0,1 is swapped with pixel W,1 where W is the width of the image.
OMX_MirrorVertical	Mirrors pixels in the vertical direction. Hence, pixel at 1,0 is swapped with pixel 1,H where H is the height of the image.
OMX_MirrorBoth	Mirrors pixels in the horizontal and vertical directions. Hence, pixel at 0, 0 is swapped with pixel W,H where W is the width of the image and H is the height of the image.

#### **4.2.174.2.18 OMX\_CONFIG\_POINTTYPE**

A point is used to specify the location of image or video data on a port relative to another source image or video stream.

OMX\_CONFIG\_POINTTYPE is defined as follows.

```
typedef struct OMX_CONFIG_POINTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nX;
    OMX_S32 nY;
} OMX_CONFIG_POINTTYPE;
```

#### **4.2.17.14.2.18.1 Parameters**

The parameters for OMX\_CONFIG\_POINTTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nX is the X-coordinate location in pixels in the horizontal direction with respect to the origin (0,0) located in the top-left corner of the image.
- nY is the Y-coordinate location in pixels in the vertical direction with respect to the origin (0,0) located in the top-left corner of the image.

#### **4.2.184.2.19 OMX\_CONFIG\_RECTTYPE**

Rectangles are used with several configuration types to indicate orientation, position, inclusion, or exclusion.

OMX\_CONFIG\_RECTTYPE is defined as follows.

```
typedef struct OMX_CONFIG_RECTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nLeft;
    OMX_S32 nTop;
    OMX_U32 nWidth;
    OMX_U32 nHeight;
} OMX_CONFIG_RECTTYPE;
```

#### **4.2.18.14.2.19.1 Parameters**

The parameters for OMX\_CONFIG\_RECTTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nLeft is the leftmost coordinate of the rectangle.
- nTop is the topmost coordinate of the rectangle.

- `nWidth` is the width of the rectangle in pixels.
- `nHeight` is the height of the rectangle in pixels.

`nLeft` and `nTop` coordinate values are with respect to the origin (0,0) located in the top-left corner of the image.

#### 4.2.19 `OMX_CONFIG_FRAMESTABTYPE`

Frame stabilization reduces motion blur during image capture or video recording. Frame stabilization is most often associated with camera sensor source components, a camera sensor filter, or a digital signal processor (DSP).

The frame stabilization feature compensates for the extremely unsteady nature of cameras on handheld devices such as a cell phone or personal digital assistant (PDA).

`OMX_CONFIG_FRAMESTABTYPE` is defined as follows.

```
typedef struct OMX_CONFIG_FRAMESTABTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bStab;
} OMX_CONFIG_FRAMESTABTYPE;
```

##### 4.2.19.1 Parameters

The parameters for `OMX_CONFIG_FRAMESTABTYPE` are defined as follows.

- `nPortIndex` is the read-only value containing the index of the port represents the port that this structure applies to.
- `bStab` is the Boolean value that enables or disables frame stabilization.

#### 4.2.20 `OMX_CONFIG_WHITEBALCONTROLTYP`

White balance control is used with camera sensors to adjust the color temperature of the image so that pure white appears as white in the image. This adjustment can be controlled automatically or manually.

`OMX_CONFIG_WHITEBALCONTROLTYP` is defined as follows.

```
typedef struct OMX_CONFIG_WHITEBALCONTROLTYP {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_WHITEBALCONTROLTYP eWhiteBalControl;
} OMX_CONFIG_WHITEBALCONTROLTYP;
```

##### 4.2.20.1 Parameters

The parameters for `OMX_CONFIG_WHITEBALCONTROLTYP` are defined as follows.

- nPortIndex is the read only value containing the index of the portrepresents the port that this structure applies to.
- eWhiteBalControl is the enumerated valued indicating the type of white balance control used. Table 4-37 details the values that can be selected for white balance control.

**Table 4-37: White Balance Control**

<b>OMX_WHITEBALCONTROTYPE Enumerated Value</b>	<b>Description</b>
OMX_WhiteBalControlOff	Disables exposure control.
OMX_WhiteBalControlAuto	Automatic white balance control. The color temperature of the captured image or video stream is adjusted per frame using a white reference from within each frame.
OMX_WhiteBalControlSunLight	Manual white balance control when the sun provides the light source.
OMX_WhiteBalControlCloudy	Manual white balance control when the sun provides the light source through clouds.
OMX_WhiteBalControlShade	Manual white balance control when the light source is the sun and the scene is in the shade.
OMX_WhiteBalControlTungsten	Manual white balance control when the light source is tungsten.
OMX_WhiteBalControlFluorescent	Manual white balance control when the light source is fluorescent.
OMX_WhiteBalControlIncandescent	Manual white balance control when the light source is incandescent.
OMX_WhiteBalControlFlash	Manual white balance control when the light source is a flash.
OMX_WhiteBalControlHorizon	Manual white balance control when the light source is the sun on the horizon.

#### 4.2.21 OMX\_CONFIG\_EXPOSURECONTROLTYPE

Exposure is used to control the image sensor exposure when capturing images or streaming video.

OMX\_CONFIG\_EXPOSURECONTROLTYPE is defined as follows.

```
typedef struct OMX_CONFIG_EXPOSURECONTROLTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_EXPOSURECONTROLTYPE eExposureControl;
} OMX_CONFIG_EXPOSURECONTROLTYPE;
```

#### 4.2.21.1 Parameters

The parameters for OMX\_CONFIG\_EXPOSURECONTROLTYPe are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- eExposureControl is an enumerated value that selects the type of exposure used. Table 4-38 details the values that can be selected for exposure.

Table 4-38: Exposure Control

OMX_EXPOSURECONTROLTYPe Enumerated Value	Description
OMX_ExposureControlOff	Disables exposure control
OMX_ExposureControlAuto	Automatic exposure
OMX_ExposureControlNight	Exposure at night
OMX_ExposureControlBackLight	Exposure with backlight illuminating the subject
OMX_ExposureControlSpotLight	Exposure with a spotlight illuminating the subject
OMX_ExposureControlSports	Exposure for sports
OMX_ExposureControlSnow	Exposure for the subject in snow
OMX_ExposureControlBeach	Exposure for the subject at a beach
OMX_ExposureControlLargeAperture	Exposure when using a large aperture on the camera
OMX_ExposureControlSmallAperture	Exposure when using a small aperture on the camera

#### 4.2.22 OMX\_CONFIG\_CONTRASTTYPe

Contrast controls the relative difference between the pixels. Contrast is applied to image or video data on the specified port.

OMX\_CONFIG\_CONTRASTTYPe is defined as follows.

```
typedef struct OMX_CONFIG_CONTRASTTYPe {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nContrast;
} OMX_CONFIG_CONTRASTTYPe;
```

#### 4.2.22.1 Parameters

The parameters for OMX\_CONFIG\_CONTRASTTYPe are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.

- nContrast is the value for contrast. The range of values is -100 to 100. The value 0x0 indicates no contrast change to pixel data.

#### **4.2.23 OMX\_CONFIG\_BRIGHTNESSTYPE**

Brightness controls the luminosity of the pixels in the video or image data. Brightness is applied to the image or video data on the specified port.

OMX\_CONFIG\_BRIGHTNESSTYPE is defined as follows.

```
typedef struct OMX_CONFIG_BRIGHTNESSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nBrightness;
} OMX_CONFIG_BRIGHTNESSTYPE;
```

##### **4.2.23.1 Parameters**

The parameters for OMX\_CONFIG\_BRIGHTNESSTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nBrightness is the value for brightness in the range 0% to 100%, where 0% produces all black pixels and 100% produces entirely white.

#### **4.2.24 OMX\_CONFIG\_BACKLIGHTTYPE**

The backlight of a flat panel type of display such as a liquid crystal display (LCD) or a thin film transistor (TFT) panel can be controlled using this configuration setting. The IL client sets the percentage brightness of the backlight and the timeout before the backlight automatically turns off.

OMX\_CONFIG\_BACKLIGHTTYPE is defined as follows.

```
typedef struct OMX_CONFIG_BACKLIGHTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nBacklight;
    OMX_U32 nTimeout;
} OMX_CONFIG_BACKLIGHTTYPE;
```

##### **4.2.24.1 Parameters**

The parameters for OMX\_CONFIG\_BACKLIGHTTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.

- nBacklight is a value that represents the backlight brightness. The range of values is 0% to 100%, where 0% is completely off and 100% is full backlight intensity.
- nTimeout is the number of milliseconds before the backlight automatically turns off. A value of 0x0 forces the backlight to remain on.

#### **4.2.25 OMX\_CONFIG\_GAMMATYPE**

Gamma is applied to the image or pixel data on the specified port to correct for the non-linear response to the brightness of pixels on a display relative to the digital value of the pixel. Gamma correction is typically applied when data is captured digitally by a camera source, or when data is shown on a display device such as a panel, CRT, or TV.

OMX\_CONFIG\_GAMMATYPE is defined as follows.

```
typedef struct OMX_CONFIG_GAMMATYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nGamma;
} OMX_CONFIG_GAMMATYPE;
```

##### **4.2.25.1 Parameters**

The parameters for OMX\_CONFIG\_GAMMATYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nGamma is the display gamma expressed in Q16 format (usually in the 2.0 to 4.0 range). The value 0 is not allowed. The details of how gamma correction is done is details of how gamma correction is done are implementation-specific.

In general, an exponential relationship between the input and output pixel intensities is assumed (i.e.  $V_{out} = V_{in}^{n\Gamma}$ ) and the gamma correction component is assumed to apply an inverse transfer function (i.e.  $V_{gamma} = V_{in}^{(1/n\Gamma)}$ ). It is also assumed that the same nGamma value applies to all three color channels.

#### **4.2.26 OMX\_CONFIG\_SATURATIONTYPE**

Saturation is applied to image or pixel data on the specified port to control the hue intensity.

OMX\_CONFIG\_SATURATIONTYPE is defined as follows.

```
typedef struct OMX_CONFIG_SATURATIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nSaturation;
} OMX_CONFIG_SATURATIONTYPE;
```

#### 4.2.26.1 Parameters

The parameters for OMX\_CONFIG\_SATURATIONTYPE are defined as follows.

- nPortIndex *is the read-only value containing the index of the portrepresents the port that this structure applies to.*
- nSaturation is the value for saturation. The range of values is -100 to 100. The value 0x0 indicates no saturation change to pixel data. A value of -100 produces all black pixels, and a value of 100 produces all white pixels.

#### 4.2.27 OMX\_CONFIG\_LIGHTNESSTYPE

Lightness is applied to image or pixel data on the specified port to control the non-linear response to the brightness of pixels.

OMX\_CONFIG\_LIGHTNESSTYPE is defined as follows.

```
typedef struct OMX_CONFIG_LIGHTNESSTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_S32 nLightness;  
} OMX_CONFIG_LIGHTNESSTYPE;
```

#### 4.2.27.1 Parameters

The parameters for OMX\_CONFIG\_LIGHTNESSTYPE are defined as follows.

- nPortIndex *is the read-only value containing the index of the portrepresents the port that this structure applies to.*
- nLightness is the value for lightness. The range of values is -100 to 100. The value 0x0 indicates no lightness change to pixel data. A value of -100 produces all black pixels, and a value of 100 produces all white pixels.

#### 4.2.28 OMX\_CONFIG\_PLANEBLENDTYPE

Plane blending is used to blend pixels from multiple sources into a single destination. The plane depth is specified such that planes with lower numbers are on top of planes with higher numbers. The blending of two planes with the same depth is undefined.

OMX\_CONFIG\_PLANEBLENDTYPE is defined as follows.

```
typedef struct OMX_CONFIG_PLANEBLENDTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nDepth;  
    OMX_U32 nAlpha;  
} OMX_CONFIG_PLANEBLENDTYPE;
```

#### **4.2.28.1 Parameters**

The parameters for OMX\_CONFIG\_PLANEBLENDTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nDepth is the depth of the plane for the port. Lower values indicate higher planes, and higher values indicate lower planes. By default, the depth value is the same as the value of nPortIndex. The nDepth is only valid when set on an input port and ignored when applied to an output port.
- nAlpha indicates the alpha value used when blending planes, if the blending operation uses global alpha. When defined on an input port, the default blending operation is  $(\text{source\_alpha} * \text{source\_color}) + ((1 - \text{source\_alpha}) * \text{destination\_color})$ , where the source is the plane associated with the config and the destination is the blended result of all lower planes. If OMX\_CONFIG\_COLOREBLENDTYPE is defined on the output port, the associated eColorBlend variable is used to determine the blending equation. For information on blending operations, see section 4.2.14. If defined on an output port, the nAlpha value is written as the per pixel alpha value in the end image (if the output format supports per pixel alpha), after performing the regular alpha calculations from the input ports if defined in combination.

#### **4.2.29 OMX CONFIG TRANSITIONEFFECTTYPE**

A component may support producing output image or video frames based on two input frames, where the sequence of the output frames forms a transition from one input frame to the next.

OMX\_CONFIG\_TRANSITIONEFFECTTYPE is defined as follows.

```
typedef struct OMX_CONFIG_TRANSITIONEFFECTTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_TRANSITIONEFFECTTYPE eEffect;  
} OMX_CONFIG_TRANSITIONEFFECTTYPE;
```

#### **4.2.29.1 Parameters**

The parameters for OMX\_CONFIG\_TRANSITIONEFFECTTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to .
- eEffect is the enumerated value indicating the transition effect to be used to generate the output frames. details the possible values for transition effects.

**Field Code Changed**

Table 4-39: eEffect Values

<u>OMX_TRANSITIONEFFECTTYPE</u> value	<u>Transition Description</u>
<u>OMX_EffectNone</u>	Used to disable or cancel the current transition effect.
<u>OMX_EffectFadeFromBlack</u>	Fades from a solid black frame to the desired input frame.
<u>OMX_EffectFadeToBlack</u>	Fades from the desired input frame to a solid black frame.
<u>OMX_EffectUnspecifiedThroughConstantColor</u>	A vendor specific effect from the first input frame to the second using a constant color frame mid transition.
<u>OMX_EffectDissolve</u>	Dissolves from the first input frame to the second.
<u>OMX_EffectWipe</u>	Wipes from the first input frame to the second.
<u>OMX_EffectUnspecifiedMixOfTwoScenes</u>	A vendor specific effect from the first input frame to the second. If multiple vendor effects are available, a random one may be chosen.

#### 4.2.294.2.30 OMX\_CONFIG\_DITHERTYPE

Dithering is used when performing color format conversion where the source color format has higher precision than the destination color format. Two standard types of dithering are supported: OMX\_DitherOrdered and OMX\_DitherErrorDiffusion. OMX\_DitherOther provides a means for vendor-specific dithering algorithms.

OMX\_CONFIG\_DITHERTYPE is defined as follows.

```
typedef struct OMX_CONFIG_DITHERTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_DITHERTYPE eDither;
} OMX_CONFIG_DITHERTYPE;
```

#### 4.2.29.14.2.30.1 Parameters

The parameters for OMX\_CONFIG\_DITHERTYPE are defined as follows.

- nPortIndex **is the read only value containing the index of the portrepresents the port that this structure applies to.**
- eDither is the type of dithering used when performing color format conversion. Table 4-40 details the values that can be selected for dithering.

**Table 4-40: Dithering Values**

OMX_DITHERTYPE Enumerated Value	Description
OMX_DitherNone	Disables dithering
OMX_DitherOrdered	Enables ordered dithering
OMX_DitherErrorDiffusion	Enables error diffusion dithering
OMX_DitherOther	Enables a vendor specific dithering algorithm

#### **4.2.304.2.31 OMX\_CONFIG\_EXPOSUREVALUETYPE**

Exposure is the amount of light which falls upon the sensor of a digital camera. Shutter speed, sensitivity, and aperture are adjusted to achieve optimal exposure of a scene. Most digital cameras offer a variety of exposure modes, from fully-automatic to semi-automatic to full manual mode.

OMX\_CONFIG\_EXPOSUREVALUETYPE is defined as follows.

```
typedef struct OMX_CONFIG_EXPOSUREVALUETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_METERINGTYPE eMetering;
    OMX_S32 xEVCompensation;
    OMX_U32 nApertureFNumber;
    OMX_BOOL bAutoAperture;
    OMX_U32 nShutterSpeedMsec;
    OMX_BOOL bAutoShutterSpeed;
    OMX_U32 nSensitivity;
    OMX_BOOL bAutoSensitivity;
} OMX_CONFIG_EXPOSUREVALUETYPE;
```

#### **4.2.30.14.2.31.1 Parameters**

The parameters for OMX\_CONFIG\_EXPOSUREVALUETYPE are defined as follows.

- nPortIndex **is the read only value containing the index of the portrepresents the port that this structure applies to.**
- eMetering is the metering type to be used. Table 4-41 lists the valid metering modes that can be used.

**Table 4-41: Metering Modes**

OMX_METERINGTYPE Enumerated Value	Description
OMX_MeteringModeAverage	Center weight average metering

<b>OMX_METERINGTYPE</b> Enumerated Value	Description
OMX_MeteringModeSpot	Spot (partial) metering
OMX_MeteringModeMatrix	Matrix or evaluative metering

- `xEVCompensation` is the Exposure Value compensation defined in Q16 format.
- `nApertureFNumber` is the aperture f-stop setting defined in Q16 format. A value of 2 implies a “f/2” setting. This setting is only valid for SetConfig if auto aperture mode is disabled (via `bAutoAperture`).not set.
- `bAutoAperture` is a Boolean value indicating if auto-aperture is to be enabled and applied.
- `nShutterSpeedMsec` is the shutter speed specified in units of milliseconds. This setting is only valid for SetConfig if auto shutter speed is disabled (via `bAutoShutterSpeed`).not set.
- `bAutoShutterSpeed` is a Boolean value indicating if auto shutter speed is to be enabled and applied.
- `nSensitivity` is the ISO sensitivity setting. A value of 100 implies a “ISO 100” setting. This setting is only valid for SetConfig if auto sensitivity is disabled (via `bAutoSensitivity`).not set.
- `bAutoSensitivity` is a Boolean value indicating if auto sensitivity is to be enabled and applied.

#### **4.2.31 OMX\_CONFIG\_CAPTUREMODETYPE**

Capture mode configuration is used to instruct the camera component how it shall behave during the course of capturing: continuous versus frame count limited capturing operations.

`OMX_CONFIG_CAPTUREMODETYPE` is defined as follows.

```
typedef struct OMX_CONFIG_CAPTUREMODETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bContinuous;
    OMX_BOOL bFrameLimited;
    OMX_U32 nFrameLimit;
} OMX_CONFIG_CAPTUREMODETYPE;
```

##### **4.2.31.1 Parameters**

The parameters for `OMX_CONFIG_CAPTUREMODETYPE` are defined as follows.

- `nPortIndex` is the read-only value containing the index of the port represents the port that this structure applies to.

- ~~bContinuous~~ is a Boolean used to indicate the frame rate emission. If true then ignore the port frame rate setting and emit captured frame data as quickly as possible otherwise obey the port's frame rate setting.
- ~~bFrameLimited~~ is a Boolean used to indicate if capturing shall be terminated after the specified number of frames if true frame limited capture is enabled; otherwise the port does not terminate capturing until instructed to do so by the client.
- ~~nFrameLimit~~ is the limit on number of frames emitted during capturing, this parameter is only valid if ~~bFrameLimited~~ is enabled.

#### **4.2.32 OMX\_CONFIG\_BOOLEANTYPE**

The OMX\_CONFIG\_BOOLEANTYPE structure contains generic Boolean configuration information that may be used to set component level configuration settings rather than port level configuration settings.

OMX\_CONFIG\_BOOLEANTYPE is defined as follows:

```
typedef struct OMX_CONFIG_BOOLEANTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_BOOL bEnabled;
} OMX_CONFIG_BOOLEANTYPE;
```

##### **4.2.32.1 Parameters**

The parameters for OMX\_CONFIG\_BOOLEANTYPE are defined as follows.

- ~~bEnabled~~ is a Boolean used to indicate if a configuration is to be enabled. The configuration setting to be enabled is typically inherent in the name of the configuration or parameter index ~~ice~~ used with this structure.

For example, the OMX\_IndexAutoPauseAfterCapture index will use the OMX\_CONFIG\_BOOLEANTYPE structure to enable or disable the auto pause mechanism after a capture request is completed.

#### **4.2.33 4.2.32 OMX\_OTHER\_EXTRADATATYPE**

The OMX\_OTHER\_EXTRADATATYPE structure is used to describe the additional buffer payload information included within the buffer. A buffer may contain multiple blocks of extra data and thus multiple instances of this structure.

Each additional EXTRADATATYPE structure shall be required to be 32 bit address aligned, and padding bytes may need to ~~be~~ inserted in order to ensure this alignment.

The order of the additional information is not required to be pre-determined since a component is expected to traverse the OMX\_OTHER\_EXTRADATATYPE structures to determine the additional information of interest.

OMX\_OTHER\_EXTRADATATYPE is defined as follows.

```

typedef struct OMX_OTHER_EXTRADATATYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_EXTRADATATYPE eType;
    OMX_U32 nDataSize;
    OMX_U8 data[1];
} OMX_OTHER_EXTRADATATYPE;

```

#### 4.2.33.14.2.32.1 Parameters

The parameters for OMX\_OTHER\_EXTRADATATYPE are defined as follows.

- nSize is the size of the structure including data bytes and any padding necessary to ensure 32bit alignment of the next OMX\_OTHER\_EXTRADATATYPE structure.
- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- eType identifies the extra data payload type. Table 4-42 details the values that can be selected for extra data payload type.

**Table 4-42: Extra Data Payload Type Enumerated values**

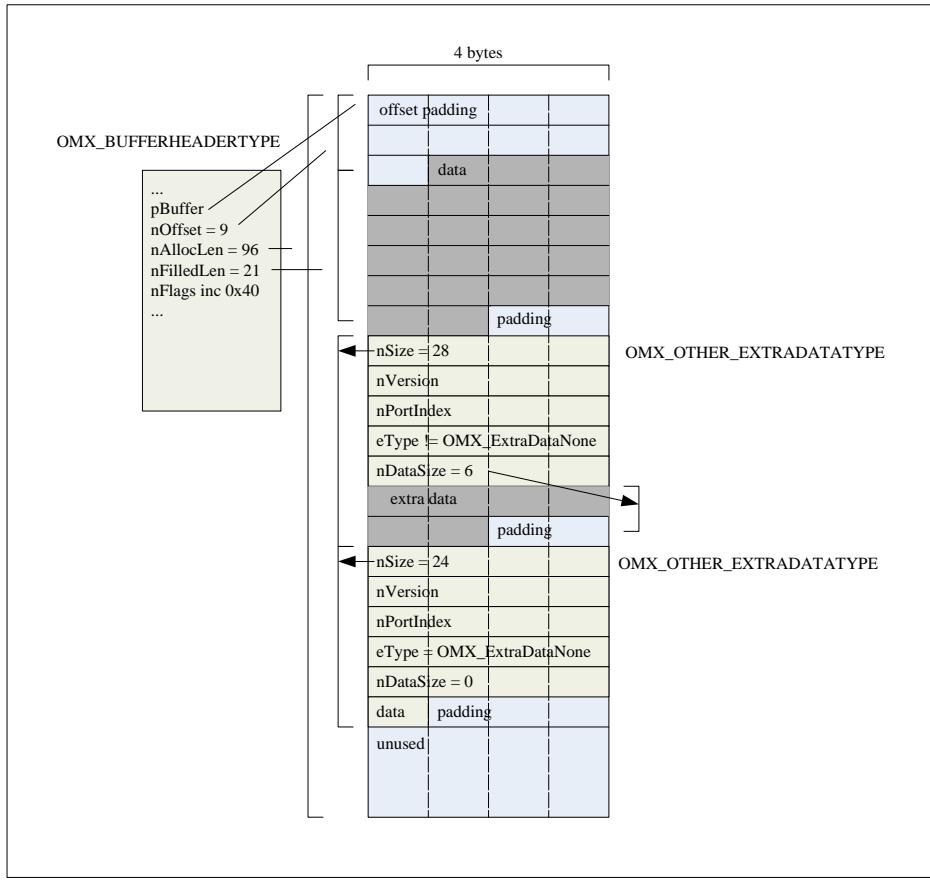
Enumerated Value	Description
OMX_ExtraDataNone	Indicates that this terminates the list of extra data sections.
OMX_ExtraDataQuantization	Indicates that the data payload contains quantization data.
<u>OMX_ExtraDataInterlaceFormat</u>	<u>Specifies the interlaced format packing of the video frame.</u> <u>The data payload contains an the interlace format associated with the payload - the interlace formats are described in Table 4-50.</u> <u>This information shall be included for all interlaced content being emitted by a port.</u>

Field Code Changed  
Field Code Changed

- nDataSize identifies the size of supporting data in units of bytes. For the OMX\_OTHER\_EXTRADATATYPE structure that terminates the list of extra data sections, nDataSize will be zero.
- data is an array of one or more bytes of data as indicated by the nDataSize field.

#### 4.2.33.24.2.32.2 Sample code

The following diagram shows the arrangement of extra data sections in a buffer.



**Figure 4-3. Formatting of Extra Buffer Data**

The following code sequence shows traversing the list of extra data sections.

```

/* Traverse the list of extra data sections */
OMX_OTHER_EXTRADATATYPE *pExtra;
OMX_U8 *pTmp = pBufferHdr->pBuffer + pBufferHdr->nOffset +
pBufferHdr->nFilledLen + 3;

pExtra = (OMX_OTHER_EXTRADATATYPE *) (((OMX_U32) pTmp) & ~3);

while(pExtra->eType != OMX_ExtraDataNone)
{
    pExtra = (OMX_OTHER_EXTRADATATYPE *) (((OMX_U8 *) pExtra) +
pExtra->nSize);
}

```

### **4.2.33 OMX\_CONFIG\_CAPTUREMODETYPE**

Capture mode configuration is used to instruct the camera component how it shall behave during the course of capturing: continuous versus frame count limited capturing operations.

OMX\_CONFIG\_CAPTUREMODETYPE is defined as follows.

```
typedef struct OMX_CONFIG_CAPTUREMODETYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BOOL bContinuous;  
    OMX_BOOL bFrameLimited;  
    OMX_U32 nFrameLimit;  
} OMX_CONFIG_CAPTUREMODETYPE;
```

#### **4.2.33.1 Parameters**

The parameters for OMX\_CONFIG\_CAPTUREMODETYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- bContinuous is a Boolean used to indicate the frame rate emission. If bContinuous is set to OMX\_TRUE, then ignore the port frame rate setting and emit captured frame data as quickly as possible otherwise obey the port's frame rate setting.
- bFrameLimited is a Boolean used to indicate whether capturing shall be terminated after the specified number of frames. If bFrameLimited is set to OMX\_TRUE, then frame limited capture is enabled; otherwise the port does not terminate capturing until instructed to do so by the client.
- nFrameLimit is the limit on number of frames emitted during capturing, this parameter is only valid if bFrameLimited is enabled.

### **4.2.34 OMX\_CONFIG\_BOOLEANTYPE**

The OMX\_CONFIG\_BOOLEANTYPE structure contains generic Boolean configuration information that may be used to set component level configuration settings rather than port level configuration settings.

OMX\_CONFIG\_BOOLEANTYPE is defined as follows.

```
typedef struct OMX_CONFIG_BOOLEANTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_BOOL bEnabled;  
} OMX_CONFIG_BOOLEANTYPE;
```

#### **4.2.34.1 Parameters**

The parameters for OMX\_CONFIG\_BOOLEANTYPE are defined as follows.

- bEnabled is a Boolean used to indicate if a configuration is to be enabled. The configuration setting to be enabled is typically inherent in the name of the configuration or parameter index used with this structure.

For example, the OMX\_IndexAutoPauseAfterCapture index will use the OMX\_CONFIG\_BOOLEANTYPE structure to enable or disable the auto pause mechanism after a capture request is completed.

#### **4.2.34 OMX\_CONFIG\_FOCUSREGIONTYPE**

OMX\_CONFIG\_FOCUSREGIONTYPE is used to define the focus region of interest.

The OMX\_CONFIG\_FOCUSREGIONTYPE can be used with OMX\_IMAGE\_CONFIG\_FOCUSCONTROLTYP to define the focus control for a specific focus region of interest.

OMX\_CONFIG\_FOCUSREGIONTYPE is defined as follows:

```
typedef struct OMX_CONFIG_FOCUSREGIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bCenter;
    OMX_BOOL bLeft;
    OMX_BOOL bRight;
    OMX_BOOL bTop;
    OMX_BOOL bBottom;
    OMX_BOOL bTopLeft;
    OMX_BOOL bTopRight;
    OMX_BOOL bBottomLeft;
    OMX_BOOL bBottomRight;
} OMX_CONFIG_FOCUSREGIONTYPE;
```

##### **4.2.34.1 Parameters**

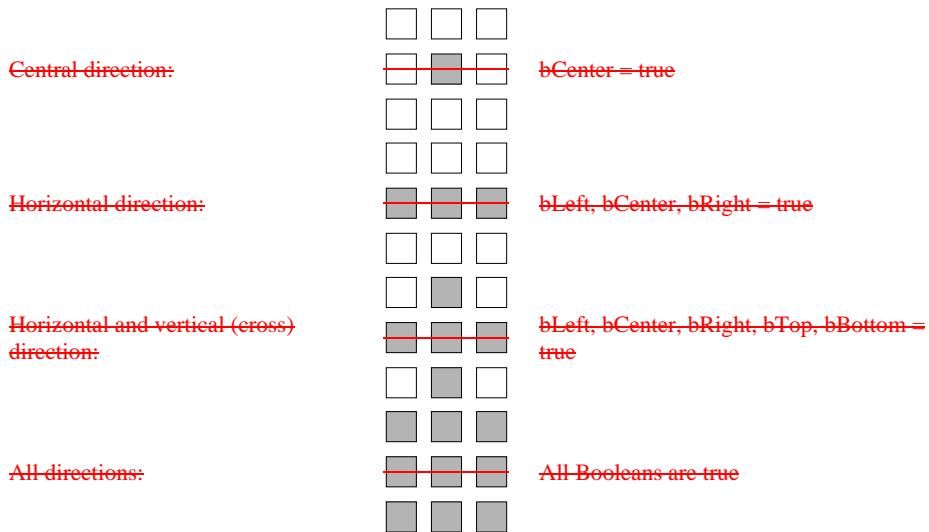
The parameters for OMX\_CONFIG\_FOCUSREGIONTYPE are defined as follows.

- nPortIndex is the read only value containing the index of the port.
- bCenter specifies if the center region is to be used as the region of interest.
- bLeft specifies if the left region is to be used as the region of interest.
- bRight specifies if the right region is to be used as the region of interest.
- bTop specifies if the top region is to be used as the region of interest.
- bBottom specifies if the bottom region is to be used as the region of interest.
- bTopLeft specifies if the top left region is to be used as the region of interest.
- bTopRight specifies if the top right region is to be used as the region of interest.
- bBottomLeft specifies if the bottom left region is to be used as the region of interest.

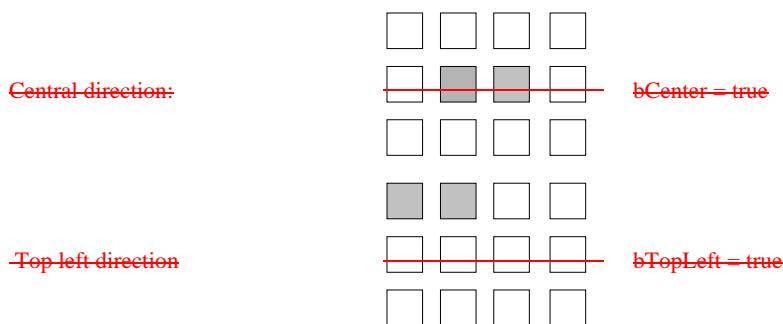
- `bBottomRight` specifies if the bottom right region is to be used as the region of interest.

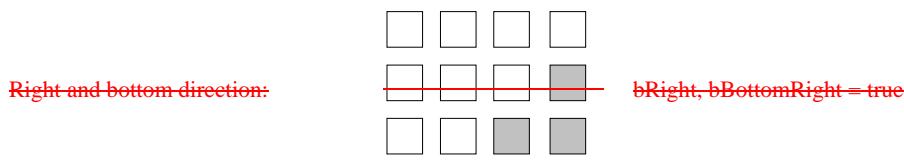
The FocusRegions should be interpreted as a direction. If more than 9 regions are available inby the hardware, the regions are mapped on the booleans above by combining regions together according implementation choice. Therefore the IL client should see the region as a focus direction.

As an example, assume there are 9 focus measurement points, 3 in horizontal and 3 in vertical direction:



As an example, assume there are 12 focus measurement points, 4 in horizontal and 3 in vertical direction:





#### 4.2.35 OMX\_PARAM\_FOCUSSTATUSTYPE

~~OMX\_PARAM\_FOCUSSTATUSTYPE~~ is used to retrieve the focus status, including detailed information on the region of interest. This structure is used in conjunction with ~~OMX\_CONFIG\_FOCUSREGIONTYPE~~.

~~OMX\_PARAM\_FOCUSSTATUSTYPE~~ is defined as follows.

```
typedef struct OMX_PARAM_FOCUSSTATUSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_FOCUSSTATUSTYPE eFocusStatus;
    OMX_BOOL bCenterStatus;
    OMX_BOOL bLeftStatus;
    OMX_BOOL bRightStatus;
    OMX_BOOL bTopStatus;
    OMX_BOOL bBottomStatus;
    OMX_BOOL bTopLeftStatus;
    OMX_BOOL bTopRightStatus;
    OMX_BOOL bBottomLeftStatus;
    OMX_BOOL bBottomRightStatus;
} OMX_PARAM_FOCUSSTATUSTYPE;
```

##### 4.2.35.1 Parameters

The parameters for ~~OMX\_CONFIG\_FOCUSREGIONTYPE~~ are defined as follows.

- ~~nPortIndex~~ is the read-only value containing the index of the port.
- ~~eFocusStatus~~ specifies the image focus status. [Table 4-43 details the possible values for image focus status.](#)

[Table 4-43: eFocus Status Types](#)

Focus Status	Focus Status Description
OMX_FocusStatusOff	Focus request is disabled
OMX_FocusStatusRequest	Focus request is currently being processed.
OMX_FocusStatusReached	Focus has been reached.
OMX_FocusStatusUnableToReach	Focus is unreachable, the maximum is too close to the average noise
OMX_FocusStatusLost	Focus has been lost, the main subject has moved in the scene

- `bCenterStatus` specifies the focus status for the center region of interest.
- `bLeftStatus` specifies the focus status for the left region of interest.
- `bRightStatus` specifies the focus status for the right region of interest.
- `bTopStatus` specifies the focus status for the top region of interest.
- `bBottomStatus` specifies the focus status for the bottom region of interest
- `bTopLeftStatus` specifies the focus status for the top left region of interest
- `bTopRightStatus` specifies the focus status for the top right region of interest
- `bBottomLeftStatus` specifies the focus status for the bottom left region of interest
- `bBottomRightStatus` specifies the focus status for the bottom right region of interest

#### **4.2.36 OMX\_CONFIG\_TRANSITIONEFFECTTYPE**

A component may support producing output image or video frames based on two input frames, where the sequence of the output frames forms a transition from one input frame to the next.

`OMX_CONFIG_TRANSITIONEFFECTTYPE` is defined as follows.

```
typedef struct OMX_CONFIG_TRANSITIONEFFECTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_TRANSITIONEFFECTTYPE eEffect;
} OMX_CONFIG_TRANSITIONEFFECTTYPE;
```

##### **4.2.36.1 Parameters**

The parameters for `OMX_CONFIG_TRANSITIONEFFECTTYPE` are defined as follows.

- `nPortIndex` is the read-only value containing the index of the output port
- `eEffect` is the enumerated value indicating the transition effect to be used to generate the output frames.

**Table : eEffect Values**

<b>OMX_TRANSITIONEFFECTTYPE value</b>	<b>Transition Description</b>
<code>OMX_EffectNone</code>	Used to disable or cancel the current transition effect.
<code>OMX_EffectFadeFromBlack</code>	Fades from a solid black frame to the desired input frame.

<b>OMX_TRANSITIONEFFECTTYPE value</b>	<b>Transition Description</b>
<code>OMX_EffectFadeToBlack</code>	Fades from the desired input frame to a solid black frame.
<code>OMX_EffectUnspecifiedThroughConstantColor</code>	A vendor specific effect from the first input frame to the second using a constant color frame mid transition.
<code>OMX_EffectDissolve</code>	Dissolves from the first input frame to the second.
<code>OMX_EffectWipe</code>	Wipes from the first input frame to the second.
<code>OMX_EffectUnspecifiedMixOfTwoScenes</code>	A vendor specific effect from the first input frame to the second. If multiple vendor effects are available, a random one may be chosen.

#### **4.2.35 OMX\_SHARPNESSTYPE**

`OMX_SHARPNESSTYPE` is used to apply differing levels of sharpness, alternatively also referred to as blurriness, in order to improve the image quality.

`OMX_SHARPNESSTYPE` is defined as follows.

```
typedef struct OMX_SHARPNESSTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nSharpness;
} OMX_SHARPNESSTYPE;
```

##### **4.2.35.1 Parameters**

The parameters for `OMX_SHARPNESSTYPE` are defined as follows.

- `nPortIndex` specifies the component port index.
- `nSharpness` is a signed value identifying the level of sharpness to apply.  
Increasing positive values applies increasing levels of sharpness while increasing negative values applies increasing levels of blur.

#### **4.2.36 OMX\_CONFIG\_ZOOMFACTORTYPE**

`OMX_CONFIG_ZOOMFACTORTYPE` is used to get and set the zoom factor value.

`OMX_CONFIG_ZOOMFACTORTYPE` is defined as follows.

```
typedef struct OMX_CONFIG_ZOOMFACTORTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BU32 xZoomFactor;
}OMX_CONFIG_ZOOMFACTORTYPE;
```

#### 4.2.36.1 Parameters

The parameters for `OMX_CONFIG_ZOOMFACTOR` are defined as follows.

- `nPortIndex` specifies the component port index.
- `xZoomFactor` is representing the zoom factor in unsigned Q16 format.  
Zooming is a method of decreasing (narrowing) the apparent field of view, and the zoom factor tells the actual value of this change.  
Zoom factor is targeted to be used both for optical and digital zoom, so each type of zoom will have its own factor. Optical zoom implies a real zoom lens, while digital zoom is accomplished electronically. In case of digital zoom, camera component first computes the largest field of view constrained by both the port aspect ratio and `OMX_IndexConfigCommonCenterFieldOfView` setting. It then applies centered cropping within this largest field of view, according to the specified digital zoom factor. The resulting cropping window has the same aspect ratio as the port. In case the requested digital zoom factor cannot be applied, the setting fails and IL client is returned error `OMX_ErrorBadParameter`.

For example, lets assume the optical zoom is fixed to a certain value. Then, `xZoomFactor` equal to 1.0 factor 1 of for digital zoom means that there is no digital zoom involved and no cropping; if in this case the digital zoom factor is increased to 2 we have a 2X (two times) digital zoom and the apparent field of view will be decreased to half (1/2) of the original in both dimensions of the original scene with same resolution.

#### 4.2.36.2 Functionality

Field of View Centering for Digital Zoom

While for the optical zoom the center of the field of view is always in the center of the observed scene, there is no similar restriction for the digital zoom. For digital zoom, the center of the field of view can be actually any point from the observed scene, or any point from camera sensor provided frames.

The coordinates of the center of the field of view will use a Q16 format representation, relative to the dimensions of the whole field of view with (0,0) being top left, and (1<<16,1<<16) being bottom right.

In case the requested center of view is off the image, the setting fails and IL client is returned error OMX\_ErrorBadParameter.

#### **4.2.37 OMX IMAGE CONFIG LOCKTYPE**

OMX\_IMAGE\_CONFIG\_LOCKTYPE is used to lock image settings.

OMX\_IMAGE\_CONFIG\_LOCKTYPE is defined as follows.

```
typedef struct OMX_IMAGE_CONFIG_LOCKTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_IMAGE_LOCKTYPE eImageLock;  
} OMX_IMAGE_CONFIG_LOCKTYPE;
```

##### **4.2.37.1 Parameters**

The parameters for OMX\_IMAGE\_CONFIG\_LOCKTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to .
- eImageLock specifies the locking mode to apply. Table 4-43 details the values that can be selected for image locking.

**Table 4-43: eImageLock Values**

<u>OMX_IMAGE_LOCKTYPE</u>	<u>Description</u>
<u>OMX_IMAGE_LockOff</u>	<u>Locking of settings shall not be applied.</u>
<u>OMX_IMAGE_LockImmediate</u>	<u>Locking of settings occurs at the end of current search iteration or immediately if there is no outstanding iteration.</u> <u>settings are frozen and no longer automatically updated.</u>
<u>OMX_IMAGE_LockAtCapture</u>	<u>Locking of settings occurs at the beginning of a capture.</u> <u>settings are frozen and no longer automatically updated.</u>  <u>For example, freezing the metering settings at the beginning of a multiple image capture.</u>

##### **4.2.37.1 Functionality**

When a setting is locked, the component shall refuse any related setting change. The IL client shall first unlock the locked setting.

The IL client is allowed to change from OMX\_IMAGE\_LockImmediate to OMX\_IMAGE\_LockAtCapture, but this change implies an implicit pass through OMX\_IMAGE\_LockOff.

#### **4.2.38 OMX\_CONFIG\_FOCUSRANGETYPE**

OMX\_CONFIG\_FOCUSRANGETYPE is used to control the range of the focus.

OMX\_CONFIG\_FOCUSRANGETYPE is defined as follows.

```
typedef struct OMX_CONFIG_FOCUSRANGETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_FOCUSRANGETYPE eFocusRange;
} OMX_CONFIG_FOCUSRANGETYPE;
```

##### **4.2.38.1 Parameters**

The parameters for OMX\_CONFIG\_FOCUSRANGETYPE are defined as follows.

- nPortIndex represents the port that this structure applies to .
- eFocusRange specifies focus range mode to apply. Table 4-44 shows the values that can be selected for focus range.

**Table 4-44: eFocusRange Values**

<u>OMX_FOCUSRANGETYPE</u>	<u>Description</u>
<u>OMX_FocusRangeAuto</u>	Allows the focus range to be chosen automatically by the component.
<u>OMX_FocusRangeHyperfocal</u>	Sets the focus range to be from half the hyperfocal distance to infinity.  Hyperfocal distance is the focus distance with the maximum depth of field. It is connected to the focal point of the lens and represents the range around that point that has acceptable sharpness.
<u>OMX_FocusRangeNormal</u>	Sets the focus range from approximately 40 cm to infinity
<u>OMX_FocusRangeSuperMacro</u>	Sets the focus range from approximately 4 to 10 cm
<u>OMX_FocusRangeMacro</u>	Sets the focus range from approximately 10 to 50 cm

<u>OMX_FOCUSRANGETYPE</u>	<u>Description</u>
<u>OMX_FocusRangeInfinity</u>	<p>Sets the focus range to be at infinity.</p> <p>The differs from OMX_FocusRangeHyperfocal in that the user is interested in objects at infinity, when everything is far away; in this case the maximum possible sharpness is achieved at infinity.</p> <p>In case of OMX_FocusRangeHyperfocal range we achieve maximum possible depth of field.</p>

### OMX IMAGE CONFIG LOCKTYPE

OMX\_IMAGE\_CONFIG\_LOCKTYPE is used to lock image settings.

OMX\_IMAGE\_CONFIG\_LOCKTYPE is defined as follows.

```
typedef struct OMX_IMAGE_CONFIG_LOCKTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_IMAGE_LOCKTYPE eImageLock;
} OMX_IMAGE_CONFIG_LOCKTYPE;
```

#### Parameters

The parameters for OMX\_IMAGE\_CONFIG\_LOCKTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- eImageLock specifies the locking mode to apply.

Table 4-46: eImageLock Values

<u>OMX_IMAGE_LOCKTYPE</u>	<u>Description</u>
<u>OMX_IMAGE_LockOff</u>	Locking of settings shall not be applied.
<u>OMX_IMAGE_LockImmediate</u>	<p>Locking of settings occurs at the end of current search iteration or immediately if there is no outstanding iteration.</p> <p>Settings are frozen and no longer automatically updated.</p>
<u>OMX_IMAGE_LockAtCapture</u>	<p>Locking of settings occurs at the beginning of a capture.</p> <p>Settings are frozen and no longer automatically updated.</p> <p>For example, freezing the metering settings at the beginning of a multiple image capture.</p>

### **Functionality**

When a setting is locked, the component shall refuse any related setting change. The IL client shall first unlock the locked setting.

The IL client is allowed to change from OMX\_IMAGE\_LockImmediate to OMX\_IMAGE\_LockAtCapture, but this change implies an implicit pass through OMX\_IMAGE\_LockOff.

#### **4.2.39 OMX IMAGE CONFIG FLASHSTATUSTYPE**

OMX\_IMAGE\_CONFIG\_FLASHSTATUSTYPE is used to query the flash status.

OMX\_IMAGE\_CONFIG\_FLASHSTATUSTYPE is defined as follows.

```
typedef struct OMX_IMAGE_CONFIG_FLASHSTATUSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_IMAGE_FLASHSTATUSTYPE eFlashStatus;
} OMX_IMAGE_CONFIG_FLASHSTATUSTYPE ;
```

##### **4.2.39.1 Parameters**

The parameters for OMX\_IMAGE\_CONFIG\_FLASHSTATUSTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to .
- eFlashStatus specifies the flash status. Table 4-45 shows the possible values for flash status.

**Table 4-45: eFlashStatus Values**

<u>OMX_IMAGE_FLASHSTATUSTYPE</u>	<u>Description</u>
<u>OMX_IMAGE_FlashUnknown</u>	<p>Flash status is unknown.</p> <p><u>Conditions under which a flash may provide this status are:</u></p> <ul style="list-style-type: none"><li>• <u>No clear state can be considered.</u></li><li>• <u>Device is in transition from one flash state to another</u></li><li>• <u>During firing of the flash since this is a short transition period.</u></li></ul>
<u>OMX_IMAGE_FlashOff</u>	Flash is off.
<u>OMX_IMAGE_FlashCharging</u>	Flash is charging.
<u>OMX_IMAGE_FlashReady</u>	Flash is ready to be fired.

<u>OMX_IMAGE_FLASHSTATUS</u>	<u>Description</u>
<u>OMX_IMAGE_FlashNotAvailable</u>	<u>Flash is not available.</u>
	<u>Flash is not present or overheated.</u>
<u>OMX_IMAGE_FlashInsufficientCharge</u>	<u>Flash cannot be charged due to insufficient battery charge.</u>

#### 4.2.40 OMX\_CONFIG\_EXTCAPTUREMODETYPE

OMX\_CONFIG\_EXTCAPTUREMODETYPE is used to configure the capture behavior. It is used for best picture selection by allowing the component to capture and emit a number of frames prior to and after the capture emission is started.

OMX\_CONFIG\_EXTCAPTUREMODETYPE is used in conjunction with OMX\_CONFIG\_CAPTUREMODETYPE.

OMX\_CONFIG\_EXTCAPTUREMODETYPE is defined as follows:

```
typedef struct OMX_CONFIG_EXTCAPTUREMODETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nFrameBefore;
    OMX_BOOL bPrepareCapture;
} OMX_CONFIG_EXTCAPTUREMODETYPE;
```

##### 4.2.40.1 Parameters

The parameters for OMX\_CONFIG\_EXTCAPTUREMODETYPE are defined as follows:

- nPortIndex represents the port that this structure applies to.
- nFrameBefore specifies how many the number of captured frames that shall be buffered up by the component after bPrepareCapture has been set to OMX\_TRUE, and while waiting for the capture bit (OMX\_IndexConfigCommonPortCapturing) to be set. These frames will be constantly updated with new captures until the capture bit is set by the client. The component will then shall emit these frames as the first frames in the multiple frame mode capture mode (defined by bFrameLimited and nFrameLimit in OMX\_CONFIG\_CAPTUREMODETYPE as defined together with OMX\_CONFIG\_EXTCAPTUREMODETYPE structure).
- bPrepareCapture specifies if the component enables pre-capturing. The component shall not deliver buffered captured frames until capturing starts.

##### 4.2.40.2 Functionality

To illustrate the functionality, sample sequences for standard multiple image capture and extended multiple capture are depicted in Figure 4-4 and Figure 4-5, respectively. In both cases it is assumed that before the first sequence call (1.0 SetConfig(...)), the

Formatted: Font: 12 pt, Not Bold

camera component and any cooperating components are instantiated and tunneled.  
Camera parameters are also set, image mode established (bOneShot has value  
OMX\_TRUE), and components are in OMX\_StateExecuting state. The  
OMX\_IndexAutoPauseAfterCapture index can be used to enable or disable the  
auto pause mechanism after the capture request is completed. The auto pause mechanism  
is disabled by default. For standard multiple image capture nFrameLimit images are  
captured after capture is triggered. For extended multiple image capture, an additional  
nFrameBefore images are captured before the capture is triggered, for a total of  
nFrameLimit+nFrameBefore images.

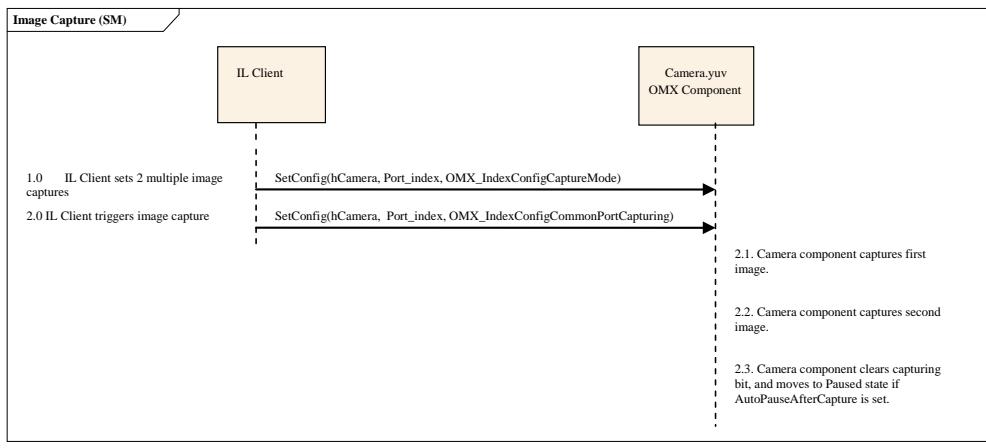
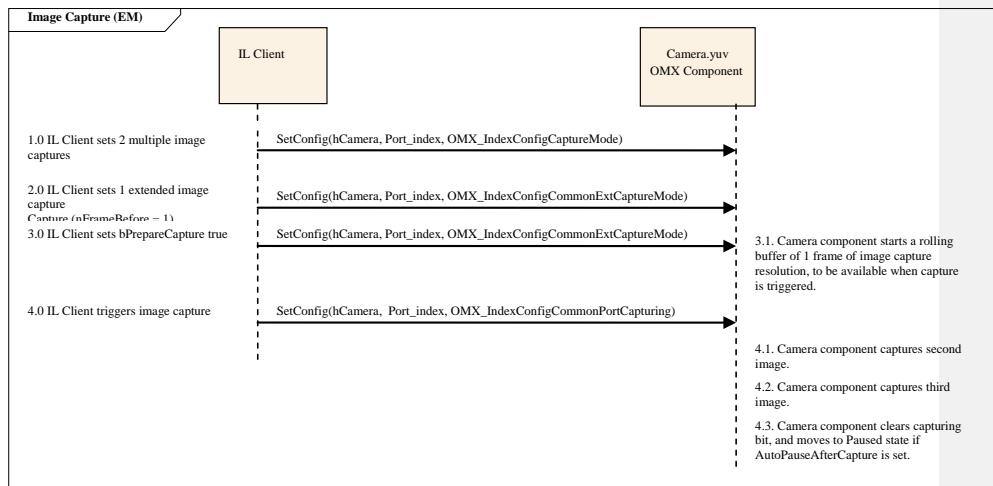


Figure 4-4: Extended Capture - Standard Multiple Image Capture



**Figure 4-5: Extended Capture - Extended Multiple Image Capture**

#### **4.2.41 OMX\_CONFIG\_NDFILTERCONTROLOTYPE**

OMX\_CONFIG\_NDFILTERCONTROLOTYPE is used to control the ND Filter functionality.

Enabling the ND filter leads to reducing the light received by the sensor, the result being that different aperture/shutter speed combinations for the same target total exposure become available.

OMX\_CONFIG\_NDFILTERCONTROLOTYPE is defined as follows.

```
typedef struct OMX_CONFIG_NDFILTERCONTROLOTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_NDFILTERCONTROLOTYPE eNDFilterControl;
} OMX_CONFIG_NDFILTERCONTROLOTYPE;
```

##### **4.2.41.1 Parameters**

The parameters for OMX\_CONFIG\_NDFILTERCONTROLOTYPE are defined as follows:

- eNDFilterControl specifies ND Filter control setting. Table 4-46 shows the possible values for ND filter settings.

**Table 4-46. eNDFilterControl Values**

<u>OMX_NDFILTERCONTROLOTYPE</u>	<u>Description</u>
<u>OMX_NDFilterOff</u>	ND Filter is off (disabled).

<u>OMX_NDFILTERCONTROLOTYPE</u>	<u>Description</u>
<u>OMX_NDFilterOn</u>	<u>ND Filter is on (enabled).</u>
<u>OMX_NDFilterAuto</u>	<u>ND Filter is on (enabled) and allows automatic control of the filter by the component.</u>

#### 4.2.42 OMX\_CONFIG\_AFASSISTANTLIGHTTYPE

OMX\_CONFIG\_AFASSISTANTLIGHTTYPE is used to control the autofocus assistant light.

OMX\_CONFIG\_AFASSISTANTLIGHTTYPE is defined as follows:

```
typedef struct OMX_CONFIG_AFASSISTANTLIGHTTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_AFASSISTANTLIGHTTYPE eAFAssistantLight;
} OMX_CONFIG_AFASSISTANTLIGHTTYPE;
```

##### 4.2.42.1 Parameters

The parameters for OMX\_CONFIG\_AFASSISTANTLIGHTTYPE are defined as follows:

- eAFAssistantLight specifies assistant light control setting. Table 4-47 shows the possible values for assistant light settings.

Table 4-47: eAFAssistantLight Values

<u>OMX_AFASSISTANTLIGHTTYPE</u>	<u>Description</u>
<u>OMX_AFAssistantLightOff</u>	<u>Forces turning off the autofocus assistant light during following autofocus cycles, and not immediately if there is an ongoing autofocus cycle.</u>
<u>OMX_AFAssistantLightOn</u>	<u>Forces turning on the autofocus assistant light during following autofocus cycles, and not immediately if there is an ongoing autofocus cycle.</u>
<u>OMX_AFAssistantLightAuto</u>	<u>Allows automatic control by the component of the autofocus assistant light during following autofocus cycles, and not immediately if there is an ongoing autofocus cycle.</u>

#### OMX\_INTERLACEFORMATTYPE

This structure is used to specify the formatting of interlaced video content.

Components such as video decoders may emit interlaced video content. When displayed on a progressive display there are visible artifacts that can be avoided using a de interlace filter. To compensate for the visible artifacts, information about the interlace format needs to be made available so that the consumer of the content may be configured appropriately.

OMX\_INTERLACEFORMATTYPE is defined as follows

```
typedef struct OMX_INTERLACEFORMATTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nFormat;  
    OMX_TICKS nTimeStamp;  
} OMX_INTERLACEFORMATTYPE;
```

### Parameters

The parameters for OMX\_INTERLACEFORMATTYPE are defined as follows.

nFormat specifies a bitmapped value identifying the interlace formats supported by the component port. This format information identifies the temporal relationship between the two fields.

The available formats are described in Table 4-49

Table 4-50: Interlace Type Values

<u>OMX_INTERLACETYPE</u> Enumerated Value	<u>Description</u>
<u>OMX_InterlaceFrameProgressive</u>	<u>The data contains a progressive (non-interlaced) frame. The data is not interlaced, it is progressive scan</u>
<u>OMX_InterlaceInterleaveFrameTopFieldFirst</u>	<u>The data contains an interlaced frame containing interleaved Top and Bottom fields.  The frame content in the buffer starts with the Top Field content as the first video line, followed by the Bottom Field video line, and the remaining video lines interleaving between Top and Bottom Field content.  The temporal relationship between the fields is that the Top Field occurs earlier than the Bottom Field.</u>

<u>OMX_InterlaceInterleaveFrameBottomFieldFirst</u>	<p>The data contains an interlaced frame containing interleaved Top and Bottom fields.</p> <p>The frame content in the buffer starts with the Top Field content as the first video line, followed by the Bottom Field video line, and the remaining video lines interleaving between Top and Bottom Field content.</p> <p>The temporal relationship between the fields is that the Bottom Field occurs earlier than the Top Field.</p>
<u>OMX_InterlaceFrameTopFieldFirst</u>	<p>The data contains an interlaced frame containing all the Top Field information followed by the Bottom Field information.</p> <p>The temporal relationship between the fields is that the Top Field occurs earlier than the Bottom Field.</p>
<u>OMX_InterlaceFrameBottomFieldFirst</u>	<p>The data contains an interlaced frame containing all the Top Field information followed by the Bottom Field information.</p> <p>The temporal relationship between the fields is that the Bottom Field occurs earlier than the Top Field.</p>

Note: In some specifications, Top\Bottom Field is referenced as Upper\Lower or Odd\Even Field.

— nTimeStamp specifies the temporal timestamp information for the second field. The nTimeStamp parameter provided via the OMX\_BUFFERHEADERTYPE structure specifies the timestamp information for the first field.

For example, for OMX\_InterlaceInterleaveFrameTopFieldFirst the nTimeStamp parameter provided via OMX\_BUFFERHEADERTYPE specifies the timestamp of the Top Field and the nTimeStamp parameter provided via this structure specifies the timestamp for the Bottom Field.

If the temporal timestamp information can not be determined for the second field, the nTimeStamp parameter for this structure shall be set the same as the nTimeStamp parameter via the OMX\_BUFFERHEADERTYPE structure.

### **OMX\_DEINTERLACETYPE**

This structure is used to enable or disable deinterlacing support. By default, deinterlacing support is disabled.

— OMX\_DEINTERLACETYPE is defined as follows

```
typedef struct OMX_DEINTERLACETYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BOOL bEnable;  
} OMX_DEINTERLACETYPE;
```

#### **Parameters**

The parameters for OMX\_DEINTERLACETYPE are defined as follows.

- bEnable specifies the requested state of the deinterlacing support.  
By default, deinterlacing is disabled.

### **OMX\_STREAMINTERLACEFORMATTYPE**

This structure is used to query if the stream contains interlaced or progressive content.

— OMX\_STREAMINTERLACEFORMATTYPE is defined as follows

```
typedef struct OMX_STREAMINTERLACEFORMATTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_BOOL bInterlaceFormat;  
    OMX_U32 nInterlaceFormats;  
} OMX_STREAMINTERLACEFORMAT;
```

#### **Parameters**

The parameters for OMX\_DEINTERLACETYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- bInterlaceFormat specifies if the stream contains interlace or progressive content — OMX\_TRUE indicates interlace and OMX\_FALSE indicates progressive.
- nInterlaceFormat specifies a bitmapped value identifying the interlace formats detected within the stream. The available formats are described in Table 4-49.  
  
It may not always be possible to determine the full extent of the formats within the stream at the commencement of the stream processing. This information may be dynamically populated as the component processes the individual frames within the stream.

### **Post-processing Conditions**

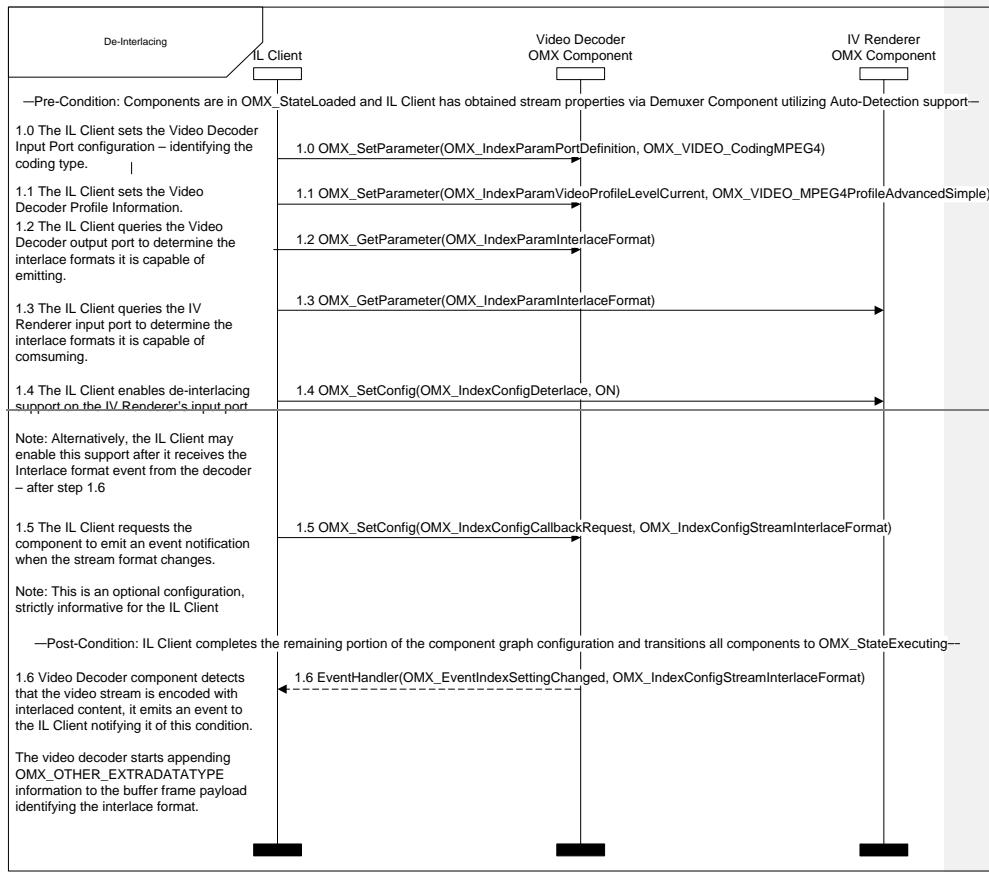
A component shall emit an OMX\_EventError event when it has detected content containing an unsupported format has been supplied to it.

### **Functionality**

This section illustrates various call sequence chart examples to configure components for interlaced video content and the notification of events.

Figure 4-6 shows the steps to coordinate the consumption of interlace content between a Video Decoder Component and IV Renderer Component.

Figure 4-7 shows the events arising when the IV Renderer component is consuming an interlace content that it does not support.



**Figure 4-6: De-Interlacing Configuration Setup**

The sequence starts with a Pre Condition that the JL Client has loaded the components, and retrieved the stream information from the Demuxer component.

The IL Client informs the Video Decoder input port that it will be consuming a MPEG4 Advanced Simple Profile stream—Step 1.0 and 1.1

The IL Client queries the Video Decoder output port to determine the possible interlace formats it may emit, this information is based on the Video Decoder's input port stream configuration – Step 1.2

The IL Client queries the IV Renderer input port to determine the possible interlace formats it can consume. Step 1.3

The IL Client enables de interlacing support on the IV Renderer input port – Step 1.4.

Note: The IL Client may wish to delay this configuration until after Step 1.6 when the Video Decoder component confirms that it will be emitting interlaced content.

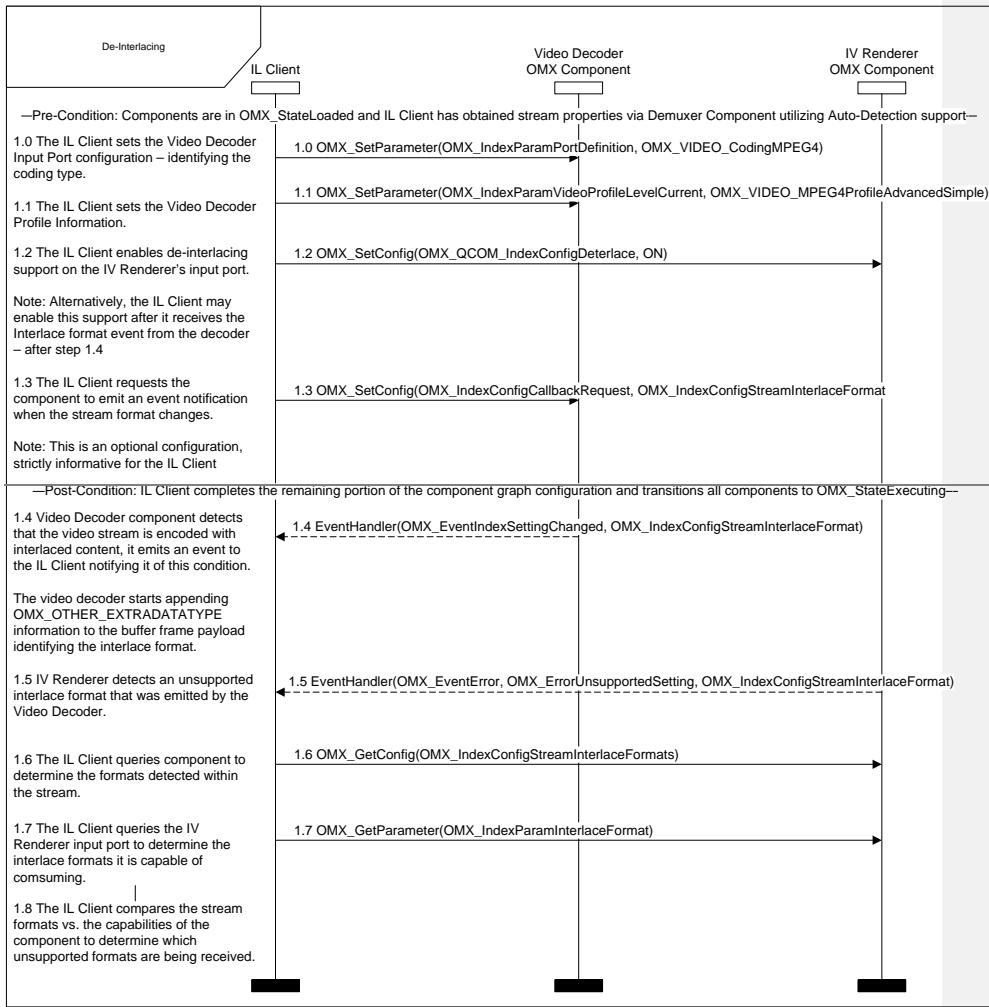
The IL Client requests the component to emit an event notification if the stream format changes from the Progressive (default mode) to Interlace – Step 1.5. This is an optional configuration, strictly informative for the IL Client.

The IL Client completes the remaining portion of the component graph configuration and transitions all the components to OMX\_StateExecuting.

At this point, the Video Decoder starts consuming the MPEG4 stream, it determines that the video stream contains interlaced content. The video decoder emits an event to the IL Client informing it of this condition and it starts appending

OMX\_OTHER\_EXTRADATATYPE information to the video frames identifying the interlace coding format – Step 1.6.

The Video Decoder output continues to emit frames.



**Figure 4-7: Unsupported Interlace Format Detection**

The sequence starts with a Pre Condition that the IL Client has loaded the components, and retrieved the stream information from the Demuxer component.

The IL Client informs the Video Decoder input port that it will be consuming a MPEG4 Advanced Simple Profile stream—Step 1.0 and 1.1

The IL Clients neglects to query both the Video Decoder output and IV Renderer input port to determine their support formats as identified in Figure 4-6 Steps 1.2 and 1.3, additionally it does not limit the type of formats to be emitted by the Video Decoder output port as identified in Figure 4-6 Step 1.4.

The outcome at this stage is that the video decoder will be emitting all possible formats that it is capable of supporting, which may be more than the formats supported by the IV Renderer input port.

The IL Client enables de interlacing support on the IV Renderer input port – Step 1.2.

Note: The IL Client may wish to delay this configuration until after Step 1.4 when the Video Decoder component confirms that it will be emitting interlaced content.

The IL Client requests the component to emit an event notification if the stream format changes from the Progressive (default mode) to Interlace – Step 1.3. This is an optional configuration, strictly informative for the IL Client.

The IL Client completes the remaining portion of the component graph configuration and transitions all the components to OMX\_StateExecuting.

At this point, the Video Decoder starts consuming the MPEG4 stream, it determines that the video stream contains interlaced content. The video decoder emits an event to the IL Client informing it of this condition and it starts appending

OMX\_OTHER\_EXTRADATATYPE information to the video frames identifying the interlace coding format – Step 1.4.

The Video Decoder output continues to emit frames.

The IV Renderer receives the emitted frames from the Video Decoder output port and determines that the embedded interlace content is not supported, it emits an OMX\_EventError to the IL Client informing it of this condition – Step 1.5.

The IV Renderer continues to render the content, some frames may be displayed showing interlacing artifacts due to the unsupported formats.

The IL Client queries the IV Renderer to determine the formats it supports and the formats it detected within the stream – Steps 1.6 and 1.7.

The IL Client compares this information to determine which unsupported format was detected by the IV Renderer and then it takes any appropriate action it deems necessary – Step 1.8.

#### **4.2.43 OMX\_FROITYPE**

OMX\_FROITYPE is used to specific regions of interest for focus.

OMX\_FROITYPE is defined as follows.

```
typedef struct OMX_FROITYPE {
    OMX_S32 nRectx;
```

```

    OMX_S32 nRectY;
    OMX_S32 nRectWidth;
    OMX_S32 nRectHeight;
    OMX_S32 xFocusDistance;
    OMX_FOCUSSTATUSTYPE eFocusStatus;
} OMX_FROITYPE;

```

#### **4.2.43.1 Parameters**

The parameters for `OMX_FROITYPE` are defined as follows.

- `nRectX` specifies the relative leftmost coordinate of a rectangle representing the region of interest. This coordinate is relative to the dimensions of the whole observed area for which the focusing is operating. All reported focusing areas shall be contained within this reference window (i.e., the reference window is represented as (0, 0, 1<<16, 1<<16)). This value is represented in Q16 format.
- `nRectY` specifies the relative topmost coordinate of a rectangle representing the region of interest. This coordinate is relative to the dimensions of the whole observed area for which the focusing is operating. All reported focusing areas shall be contained within this reference window (i.e., the reference window is represented as (0, 0, 1<<16, 1<<16)). This value is represented in Q16 format.
- `nRectWidth` specifies the relative width of a rectangle representing the region of interest. This coordinate is relative to the dimensions of the whole observed area for which the focusing is operating. All reported focusing areas shall be contained within this reference window (i.e., the reference window is represented as (0, 0, 1<<16, 1<<16)). This value is represented in Q16 format.
- `nRectHeight` specifies the relative height of a rectangle representing the region of interest. This coordinate is relative to the dimensions of the whole observed area for which the focusing is operating. All reported focusing areas shall be contained within this reference window (i.e., the reference window is represented as (0, 0, 1<<16, 1<<16)). This value is represented in Q16 format.
- `xFocusDistance` is the estimated focusing distance in meters. This value is represented in Q16 format. When `xFocusDistance` takes on the represented number is the maximum Q16 value, it means the distance cannot be reported since AF is at infinity. When `xFocusDistance` is 0, it means the distance is unknown. This value is represented in Q16 format.
- `eFocusStatus` specifies the status of the focus. -Table 4-49 details the possible values for focus status.

**Table 4-48 : eFocus Status Types**

<b>Focus Status</b>	<b>Focus Status Description</b>
<code>OMX_FocusStatusOff</code>	Focus request is disabled
<code>OMX_FocusStatusRequest</code>	Focus request is currently being processed.

<b>Focus Status</b>	<b>Focus Status Description</b>
<u>OMX_FocusStatusReached</u>	Focus has been reached.
<u>OMX_FocusStatusUnableToReach</u>	Focus is unreachable, the maximum is too close to the average noise
<u>OMX_FocusStatusLost</u>	Focus has been lost, the main subject has moved in the scene

<b>Focus Status</b>	<b>Focus Status Description</b>
<u>OMX_FocusStatusOff</u>	Focus request is disabled
<u>OMX_FocusStatusRequest</u>	Focus request is currently being processed.
<u>OMX_FocusStatusReached</u>	Focus has been reached.
<u>OMX_FocusStatusUnableToReach</u>	Focus is unreachable, the maximum is too close to the average noise
<u>OMX_FocusStatusLost</u>	Focus has been lost, the main subject has moved in the scene

#### **4.2.44 OMX CONFIG FOCUSREGIONSTATUSTYPE**

OMX\_CONFIG\_FOCUSREGIONSTATUSTYPE is used to retrieve the status of the focus.

OMX\_CONFIG\_FOCUSREGIONSTATUSTYPE is defined as follows.

```
typedef struct OMX_CONFIG_FOCUSREGIONSTATUSTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_BOOL bFocused;
    OMX_U32 nMaxFAreas;
    OMX_U32 nFAreas;
    OMX_FROITYPE sFROIs[1];
} OMX_CONFIG_FOCUSREGIONSTATUSTYPE;
```

##### **4.2.44.1 Parameters**

The parameters for OMX\_CONFIG\_FOCUSREGIONSTATUSTYPE are defined as follows.

- bFocused is true when any of the focus regions is currently successfully focused.

- nMaxFAreas is the maximum number of focusing areas that are to be reported. When IL client queries this information with the value set to zero, the component updates the value with the maximum supported number of focus regions and no other fields are updated. If the IL client queries this information with this value set to non-zero and within the range of supported values, the component does not update the value and only reports focus regions up to this number; so in this case the parameter is the size of returned array sFROIs. If the IL client calls GetConfig with this value non-zero and outside the range of supported values the IL client is returned error OMX\_ErrorBadParameter.
- nFAreas is the actual number of regions used by focusing. By default, this value is 1.
- sFROIs is an array which contains the coordinates of the areas in focus.

#### **4.2.44.2 Usage**

If there are multiple focus regions, the indication that focus has been achieved is updated if any of the focus regions is able to achieve focus.

The IL client needs to subscribe to callbacks in order to get events when focus status changes.

#### **4.2.45 OMX MANUALFOCUSRECTTYPE**

OMX\_MANUALFOCUSRECTTYPE is used to indicate the manual focus rectangle information.

OMX\_MANUALFOCUSRECTTYPE is defined as follows.

```
typedef struct OMX_MANUALFOCUSRECTTYPE {
    OMX_S32 nRectX;
    OMX_S32 nRectY;
    OMX_S32 nRectWidth;
    OMX_S32 nRectHeight;
} OMX_MANUALFOCUSRECTTYPE;
```

#### **4.2.45.1 Parameters**

The parameters for OMX\_MANUALFOCUSRECTTYPE are defined as follows.

- nRectX specifies the leftmost coordinate of the rectangle. These coordinates are relative to the resolution of the port. All focusing areas that are specified by the IL client shall be contained within this reference window. This value is represented in Q16 format.
- nRectY specifies the topmost coordinate of the rectangle. All focusing areas that are specified by the IL client shall be contained within this reference window. This value is represented in Q16 format.

- nRectWidth specifies the width of the rectangle. All focusing areas that are specified by the IL client shall be contained within this reference window. This value is represented in Q16 format.
- nRectHeight specifies the height of the rectangle. All focusing areas that are specified by the IL client shall be contained within this reference window. This value is represented in Q16 format.

#### **4.2.46 OMX\_CONFIG\_FOCUSREGIONCONTROLTYPe**

OMX\_CONFIG\_FOCUSREGIONCONTROLTYPe is used to set and get the control information for focus regions.

OMX\_CONFIG\_FOCUSREGIONCONTROLTYPe is defined as follows.

```
typedef struct OMX_CONFIG_FOCUSREGIONCONTROLTYPe {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nFAreas;
    OMX_FOCUSREGIONCONTROLTYPe eFocusRegionsControl;
    OMX_MANUALFOCUSRECTTYPe sManualFRegions[1];
} OMX_CONFIG_FOCUSREGIONCONTROLTYPe;
```

##### **4.2.46.1 Parameters**

The parameters for OMX\_CONFIG\_FOCUSREGIONCONTROLTYPe are defined as follows.

- nFAreas is the number of regions to be used for focusing. By default, this value is 1.

While in the case of manual focus OMX\_FocusRegionControlManual, the value of nFAreas represents the dimension of the sManualFRegions vector. In the non-manual case, the value of nFAreas represents the maximum number of focus regions to be used for focusing. When queried in manual focus cases only the required number of regions are reported.

This value limits the number of areas where focus is attempted and it is reported in the focus status OMX\_CONFIG\_FOCUSREGIONSTATUSTYPE. The maximum supported number of areas can be queried via OMX\_CONFIG\_FOCUSREGIONSTATUSTYPE (Refer to 4.2.44)

- eFocusRegionsControl specifies the focusing control type. Table 4-49 shows the possible values for focus control type.

**Table 4-49 : eFocusRegionControl Values**

<u>OMX_FOCUSREGIONCONTROLTYP</u>	<u>Description</u>
<u>OMX_FocusRegionControlAuto</u>	<u>Used when focus regions are automatically selected by autofocus (AF) algorithm.</u>
<u>OMX_FocusRegionControlManual</u>	<u>Used when focus regions are manually selected by the IL client.</u>
<u>OMX_FocusRegionControlFacePriority</u>	<u>Used when focus should be attempted to priority face if available, otherwise automatically selected by AF algorithm.</u>
<u>OMX_FocusRegionControlObjectPriority</u>	<u>Used when focus should be attempted to priority object if available, otherwise automatically selected by AF algorithm.</u>

- sManualFRegions is an array which contains the relative coordinates to be used for focusing (top left corner coordinates and size).  
This information is provided by the IL client; it is used only when eFocusRegionsControl is set to OMX\_FocusRegionControlManual.

#### 4.2.47 OMX\_INTERLACEFORMATTYPE

This structure is used to specify the formatting of interlaced video content.

Components such as video decoders may emit interlaced video content. When displayed on a progressive display there are visible artifacts that can be avoided using a de-interlace filter. To compensate for the visible artifacts, information about the interlace format needs to be made available so that the consumer of the content may be configured appropriately.

OMX\_INTERLACEFORMATTYPE is defined as follows

```
typedef struct OMX_INTERLACEFORMATTYPE {
```

```

OMX_U32 nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_U32 nFormat;
OMX_TICKS nTimeStamp;
} OMX_INTERLACEFORMATTYPE;

```

#### 4.2.47.1 Parameters

The parameters for `OMX_INTERLACEFORMATTYPE` are defined as follows.

- `nFormat` specifies a bitmapped value identifying the interlace formats supported by the component port. This format information identifies the temporal relationship between the two fields.

The available formats are described in

Field Code Changed

**Table 4-50: Interlace Type Values**

<u>OMX_INTERLACETYPE</u> Enumerated Value	<u>Description</u>
<u>OMX_InterlaceFrameProgressive</u>	<p>The data contains a progressive (non-interlaced) frame.</p> <p>The data is not interlaced, it is progressive scan</p>
<u>OMX_InterlaceInterleaveFrameTopFieldFirst</u>	<p>The data contains an interlaced frame containing interleaved Top and Bottom fields.</p> <p>The frame content in the buffer starts with the Top Field content as the first video line, followed by the Bottom Field video line, and the remaining video lines interleaving between Top and Bottom Field content.</p> <p>The temporal relationship between the fields is that the Top Field occurs earlier than the Bottom Field.</p>

<u>OMX_InterlaceInterleaveFrameBottomFieldFirst</u>	<p>The data contains an interlaced frame containing interleaved Top and Bottom fields.</p> <p>The frame content in the buffer starts with the Top Field content as the first video line, followed by the Bottom Field video line, and the remaining video lines interleaving between Top and Bottom Field content.</p> <p>The temporal relationship between the fields is that the Bottom Field occurs earlier than the Top Field.</p>
<u>OMX_InterlaceFrameTopFieldFirst</u>	<p>The data contains an interlaced frame containing all the Top Field information followed by the Bottom Field information.</p> <p>The temporal relationship between the fields is that the Top Field occurs earlier than the Bottom Field.</p>
<u>OMX_InterlaceFrameBottomFieldFirst</u>	<p>The data contains an interlaced frame containing all the Top Field information followed by the Bottom Field information.</p> <p>The temporal relationship between the fields is that the Bottom Field occurs earlier than the Top Field.</p>
<u>OMX_InterlaceInterleaveFieldTop</u>	<p>The Data contains an interlaced frame containing a single field of interlaced content. The data contains only the Top field information, with the field content occupying every other video line starting from the first line in the buffer.</p>
<u>OMX_InterlaceInterleaveFieldBottom</u>	<p>The Data contains an interlaced frame containing a single field of interlaced content. The data contains only the Bottom field information, with the field content occupying every other video line starting from the second line in the buffer.</p>

Note: In some specifications, Top\Bottom Field is referenced as Upper\Lower or Odd\Even Field.

- nTimeStamp specifies the temporal timestamp information for the second field.  
The nTimeStamp parameter provided via the OMX\_BUFFERHEADERTYPE structure specifies the timestamp information for the first field.

For example, for OMX\_InterlaceInterleaveFrameTopFieldFirst the nTimeStamp parameter provided via OMX\_BUFFERHEADERTYPE specifies the timestamp of the Top Field and the nTimeStamp parameter provided via OMX\_INTERLACEFORMATTYPE structure specifies the timestamp for the Bottom Field.

If the temporal timestamp information can not be determined for the second field, the nTimeStamp parameter for OMX\_INTERLACEFORMATTYPE structure shall be set the same as the nTimeStamp parameter via the OMX\_BUFFERHEADERTYPE structure.

#### **4.2.48 OMX\_DEINTERLACETYPE**

This structure is used to enable or disable deinterlacing support. By default, deinterlacing support is disabled.

OMX\_DEINTERLACETYPE is defined as follows

```
typedef struct OMX_DEINTERLACETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnable;
} OMX_DEINTERLACETYPE;
```

##### **4.2.48.1 Parameters**

The parameters for OMX\_DEINTERLACETYPE are defined as follows.

- eEnable specifies the requested state of the deinterlacing support.  
By default, deinterlacing is disabled.

#### **4.2.49 OMX\_STREAMINTERLACEFORMATTYPE**

This structure is used to query if the stream contains interlaced or progressive content.

OMX\_STREAMINTERLACEFORMATTYPE is defined as follows

```
typedef struct OMX_STREAMINTERLACEFORMATTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bInterlaceFormat;
    OMX_U32 nInterlaceFormats;
} OMX_STREAMINTERLACEFORMAT;
```

#### **4.2.49.1 Parameters**

The parameters for `OMX_DEINTERLACETYPE` are defined as follows.

- `nPortIndex` represents the port that this structure applies to.
- `bInterlaceFormat` specifies if the stream contains interlace or progressive content – `OMX_TRUE` indicates interlace and `OMX_FALSE` indicates progressive.
- `nInterlaceFormat` specifies a bitmapped value identifying the interlace formats detected within the stream. The available formats are described in [Table 4-1](#).  
*It may not always be possible to determine the full extent of the formats within the stream at the commencement of the stream processing. This information may be dynamically populated as the component processes the individual frames within the stream.*

**Field Code Changed**

#### **4.2.49.2 Post-processing Conditions**

A component shall emit an `OMX_EventError` event when it has detected content containing an unsupported format has been supplied to it.

#### **4.2.49.3 Functionality**

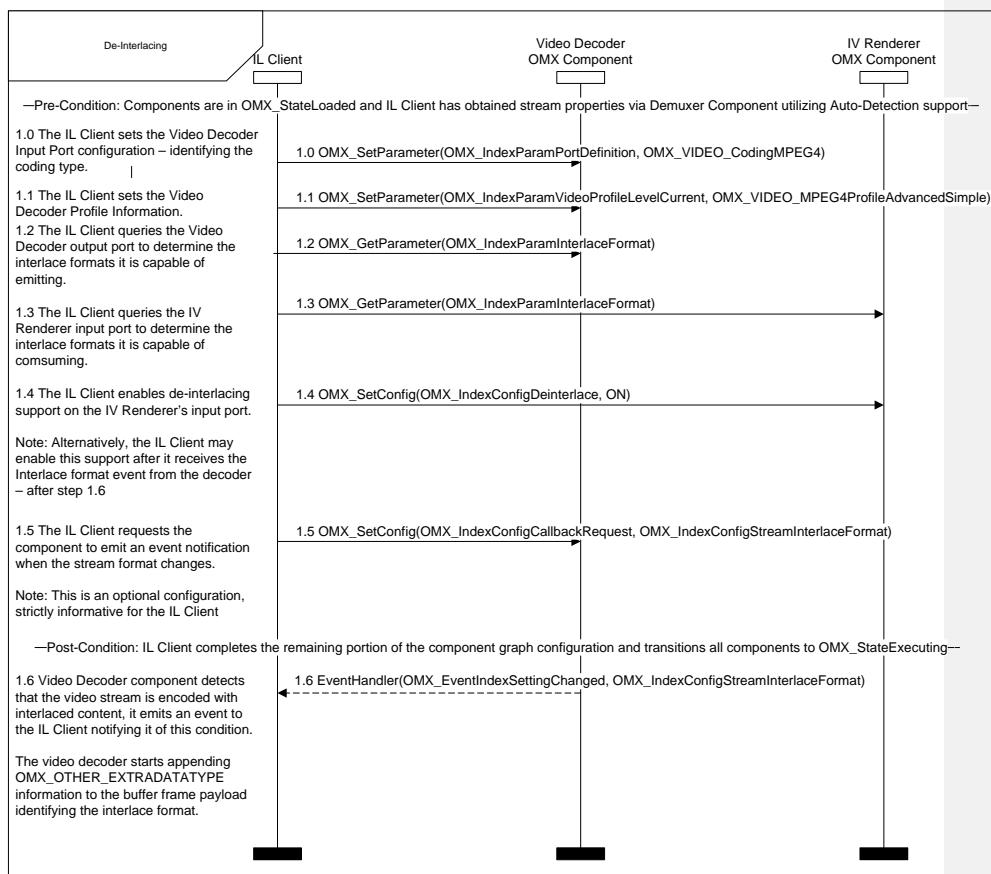
This section illustrates various call sequence chart examples to configure components for interlaced video content and the notification of events.

[Figure 4-6](#) shows the steps to coordinate the consumption of interlace content between a Video Decoder Component and IV Renderer Component.

**Field Code Changed**

[Figure 4-7](#) shows the events arising when the IV Renderer component is consuming an interlace content that it does not support.

**Field Code Changed**



**Figure 4-6: De-Interlacing Configuration Setup**

The sequence starts with a Pre-Condition that the **IL client** has loaded the components, and retrieved the stream information from the Demuxer component.

The **IL client** informs the Video Decoder input port that it will be consuming a MPEG4 Advanced Simple Profile stream – Step 1.0 and 1.1

The **IL client** queries the Video Decoder output port to determine the possible interlace formats it may emit, this information is based on the Video Decoder's input port stream configuration – Step 1.2.

The **IL client** queries the IV Renderer input port to determine the possible interlace formats it can consume – Step 1.3.

The **IL client** enables de-interlacing support on the IV Renderer input port – Step 1.4.

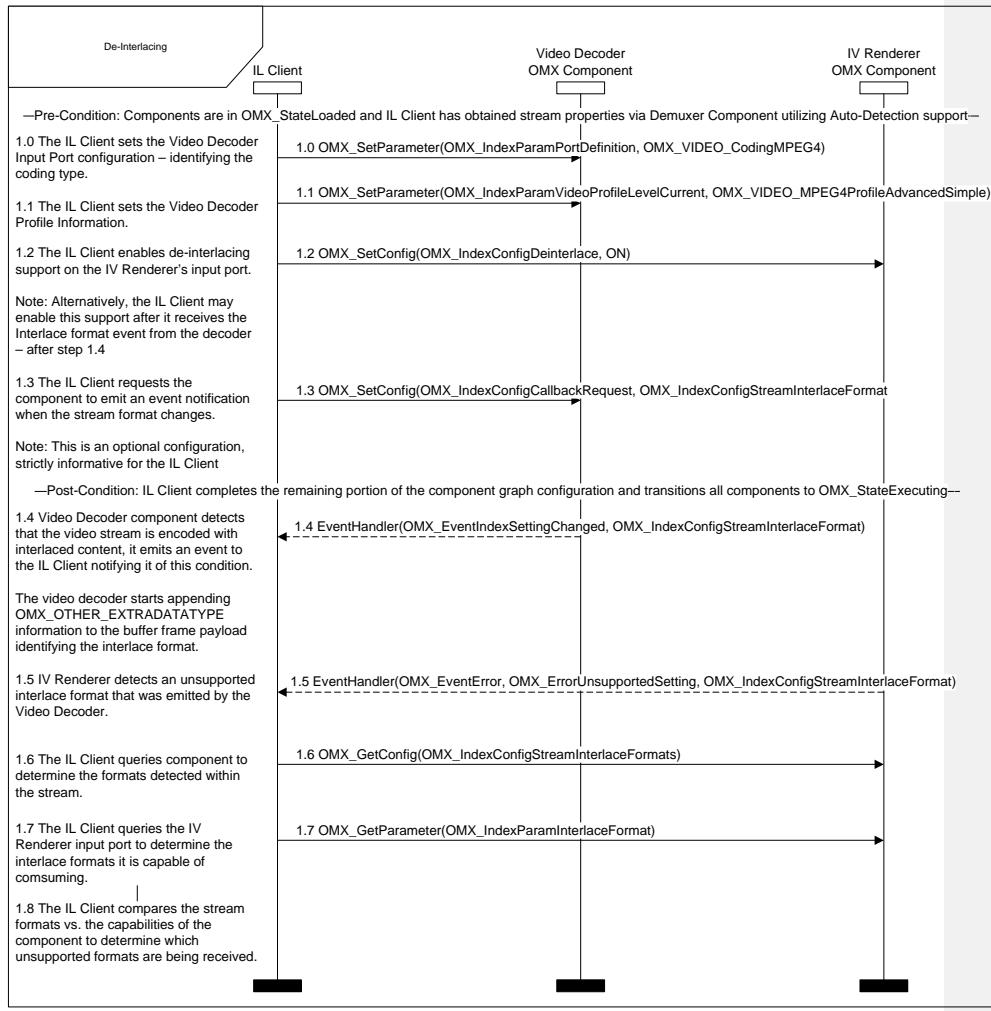
Note: The **IL client** may wish to delay this configuration until after Step 1.6 when the Video Decoder component confirms that it will be emitting interlaced content.

The **IL client** requests the component to emit an event notification if the stream format changes from the Progressive (default mode) to Interlace – Step 1.5. This is an optional configuration, strictly informative for the **IL client**.

The **IL client** completes the remaining portion of the component graph configuration and transitions all the components to OMX\_StateExecuting.

At this point, the Video Decoder starts consuming the MPEG4 stream, it determines that the video stream contains interlaced content. The video decoder emits an event to the **IL client** informing it of this condition and it starts appending OMX\_OTHER\_EXTRADATATYPE information to the video frames identifying the interlace coding format – Step 1.6.

The Video Decoder output continues to emit frames.



**Figure 4-7: Unsupported Interlace Format Detection**

The sequence starts with a Pre-Condition that the **IL client** has loaded the components, and retrieved the stream information from the Demuxer component.

The **IL client** informs the Video Decoder input port that it will be consuming a MPEG4 Advanced Simple Profile stream – Step 1.0 and 1.1

The **IL clients** neglects to query both the Video Decoder output and IV Renderer input port to determine their support formats as identified in Steps 1.2 and 1.3, additionally it does not limit the type of formats to be emitted by the Video Decoder output port as identified in Step 1.4.

Field Code Changed

The outcome at this stage is that the video decoder will be emitting all possible formats that it is capable of supporting, which may be more than the formats supported by the IV Renderer input port.

Field Code Changed

The **IL client** enables de-interlacing support on the IV Renderer input port – Step 1.2.

Note: The **IL client** may wish to delay this configuration until after Step 1.4 when the Video Decoder component confirms that it will be emitting interlaced content.

The **IL client** requests the component to emit an event notification if the stream format changes from the Progressive (default mode) to Interlace – Step 1.3. This is an optional configuration, strictly informative for the **IL client**.

The **IL client** completes the remaining portion of the component graph configuration and transitions all the components to `OMX_StateExecuting`.

At this point, the Video Decoder starts consuming the MPEG4 stream, it determines that the video stream contains interlaced content. The video decoder emits an event to the **IL client** informing it of this condition and it starts appending `OMX_OTHER_EXTRADATATYPE` information to the video frames identifying the interlace coding format – Step 1.4.

The Video Decoder output continues to emit frames.

The IV Renderer receives the emitted frames from the Video Decoder output port and determines that the embedded interlace content is not supported, it emits an `OMX_EventError` to the **IL client** informing it of this condition – Step 1.5.

The IV Renderer continues to render the content, some frames may be displayed showing interlacing artifacts due to the unsupported formats.

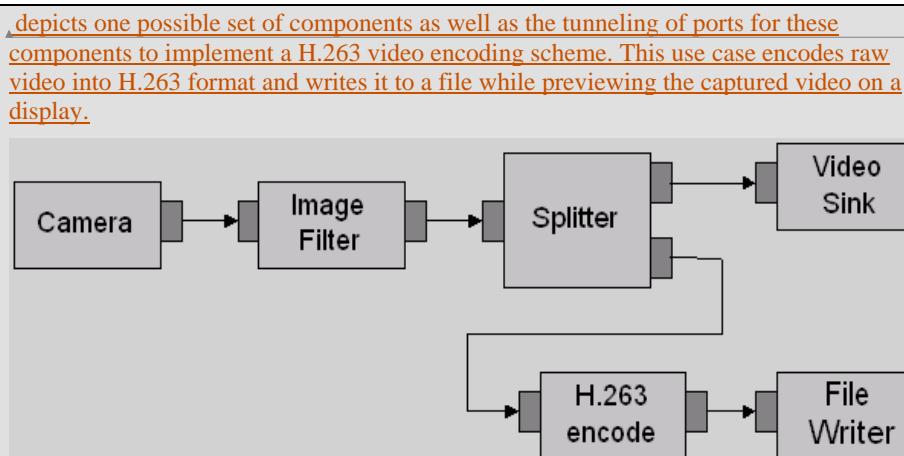
The **IL client** queries the IV Renderer to determine the formats it supports and the formats it detected within the stream – Steps 1.6 and 1.7.

The **IL client** compares this information to determine which unsupported format was detected by the IV Renderer and then it takes any appropriate action it deems necessary – Step 1.8.

## 4.3 Video

This section describes the parameter and configuration details for ports in the video domain. These parameter and configuration details are specified in the OMX\_Video.h header.

### 4.3.1 Video Use Case Examples



Field Code Changed

Figure 4-8. H.263 Video Encode Use Case

shows six components, namely the camera, the image filter, the splitter, the H.263 video encoder, the file writer, and the video sink.

Field Code Changed

shows a more complex use case, which is video conferencing. This use case supports simultaneous encoding and decoding of video streams. To simplify the use case, the corresponding audio components are not included.

Field Code Changed

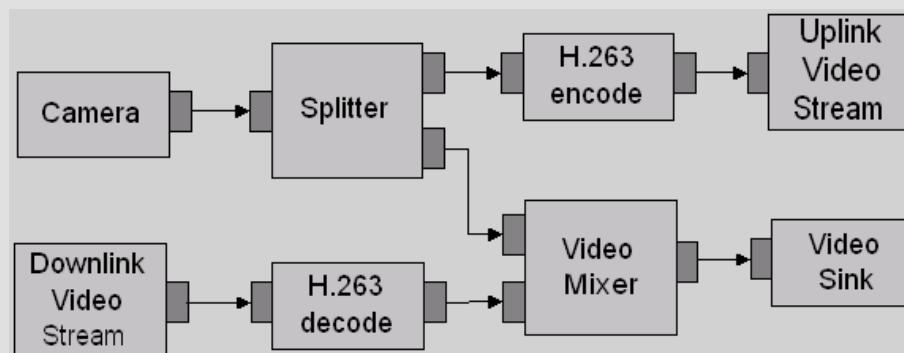


Figure 4-9. Video Conferencing Use Case

Raw video is encoded to H.263 format and then transmitted via a video uplink to the far-side conferencing participant. At the same time, a H.263 video stream is received from

the far-side participant via a video downlink and decoded to raw video format before being mixed into a pre-determined presentation layout via the video mixer such that both the local participant's video and far-side participant's video are displayed via the local video sink.

#### **4.3.14.3.2 General Enumerations**

The OMX\_VIDEO\_CODINGTYPE enumeration defines the video coding types supported.. If OMX\_VIDEO\_CodingUnused is selected, then the coding selection shall be done in a vendor-specific way. Table 4-51 shows the OpenMAX IL-supported video compression formats.

**Table 4-51: Supported Video Compression Formats**

Field Name	Coding Type Descriptions	References to Standards
OMX_VIDEO_CodingUnused	No coding applied. Use eColorFormat	Not available
OMX_VIDEO_CodingAutoDetect	Auto-detection by the OpenMAX IL component	Not available
OMX_VIDEO_CodingMPEG2	MPEG-2, also known as H.262 video format	<a href="#">MPEG2</a>
OMX_VIDEO_CodingH263	ITU H.263 video format	<a href="#">H263</a>
OMX_VIDEO_CodingMPEG4	MPEG-4 video format	<a href="#">MPEG4</a>
OMX_VIDEO_CodingWMV	All versions of the Windows Media video format	<a href="#">WMV</a>
OMX_VIDEO_CodingRV	All versions of the RealVideo® format	<a href="#">RV</a>
OMX_VIDEO_CodingAVC	ITU H.264/AVC video format	<a href="#">H264</a>
OMX_VIDEO_CodingMJPEG	Motion JPEG video format	<a href="#">MJPEG</a>
<u>OMX_VIDEO_CodingVC1</u>	<u>VC-1 format (SMPTE 421M)</u>	<u>VC-1</u>
<u>OMX_VIDEO_CodingVP8</u>	<u>VP8 Video Bitstream</u>	<u>VP8</u>
<u>OMX_VIDEO_CodingMax</u>	<u>Maximum value</u>	<u>N/A</u>

The OMX\_VIDEO\_PICTURETYPE enumeration defines the video picture types supported. Table 4-52 describes the supported video picture types.

**Table 4-52: Supported Video Picture Types**

Field Name	Picture Type Descriptions
OMX_VIDEO_PictureTypeI	General I-frame type
OMX_VIDEO_PictureTypeP	General P-frame type
OMX_VIDEO_PictureTypeB	General B-frame type
OMX_VIDEO_PictureTypeSI	H.263 SI-frame type

Field Name	Picture Type Descriptions
OMX_VIDEO_PictureTypeSP	H.263 SP-frame type
OMX_VIDEO_PictureTypeEI	H.264 EI-frame type
OMX_VIDEO_PictureTypeEP	H.264 EP-frame type
OMX_VIDEO_PictureTypeS	MPEG-4 S-frame type
<b>OMX_VIDEO_PictureTypeMax</b>	<b>Maximum value</b>

#### **4.3.24.3.3 Parameter and Configuration Indices**

The header OMX\_Index.h contains the enumeration OMX\_INDEXTYPE, which contains all of the standard index values used with the OpenMAX IL core functions OMX\_GetParameter, OMX\_SetParameter, OMX\_GetConfig, and OMX\_SetConfig.

The index values that relate to video are described in this section. For example, OMX\_IndexParamVideoPortFormat index is used with OMX\_GetParameter and OMX\_SetParameter to access the OMX\_VIDEO\_PARAM\_PORTFORMATTYPE. Table 4-53 identifies the video indices.

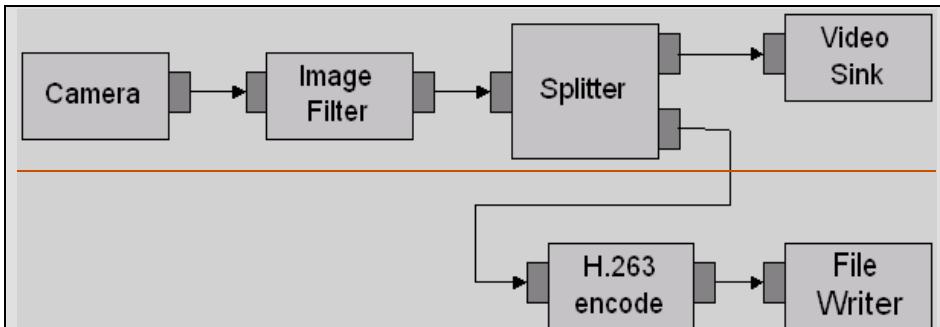
**Table 4-53: Video Indices**

OpenMAX IL Indices ( OMX_Index.h )	Corresponding OpenMAX IL Video Structures ( OMX_Video.h )
<b>OMX_IndexParamPortDefinition</b>	<b>OMX_PARAM_PORTDEFINITIONTYPE with OMX_VIDEO_PORTDEFINITIONTYPE</b>
OMX_IndexParamVideoPortFormat	OMX_VIDEO_PARAM_PORTFORMATTYPE
OMX_IndexParamVideoQuantization	OMX_VIDEO_PARAM_QUANTIZATIONTYPE
OMX_IndexParamVideoFastUpdate	OMX_VIDEO_PARAM_VIDEOFASTUPDATETYPE
OMX_IndexParamVideoBitrate	OMX_VIDEO_PARAM_BITRATETYPE
OMX_IndexParamVideoMotionVector	OMX_VIDEO_PARAM_MOTIONVECTORTYPE
OMX_IndexParamVideoIntraRefresh	OMX_VIDEO_PARAM_INTRAREFRESHTYPE
<b>OMX_IndexConfigVideoIntraRefresh</b>	<b>OMX_VIDEO_PARAM_INTRAREFRESHTYPE</b>
OMX_IndexParamVideoErrorCorrection	OMX_VIDEO_PARAM_ERRORCORRECTIONTYPE
OMX_IndexParamVideoVBSMC	OMX_VIDEO_PARAM_VBSMCTYPE
OMX_IndexParamVideoMpeg2	OMX_VIDEO_PARAM_MPEG2TYPE
OMX_IndexParamVideoMpeg4	OMX_VIDEO_PARAM_MPEG4TYPE
OMX_IndexParamVideoWmv	OMX_VIDEO_PARAM_WMVTYPE
OMX_IndexParamVideoRv	OMX_VIDEO_PARAM_RVTYPE
OMX_IndexParamVideoAvc	OMX_VIDEO_PARAM_AVCTYPE
OMX_IndexParamVideoH263	OMX_VIDEO_PARAM_H263TYPE
<b>OMX_IndexParamVideoVp8</b>	<b>OMX_VIDEO_PARAM_VP8TYPE</b>
<b>OMX_IndexConfigVideoVp8ReferenceFrame</b>	<b>OMX_VIDEO_VP8REFERENCEFRAMETYPE</b>

OpenMAX IL Indices ( <a href="#">OMX_Index.h</a> )	Corresponding OpenMAX IL Video Structures ( <a href="#">OMX_Video.h</a> )
<a href="#"><u>OMX_IndexConfigVideoVp8ReferenceFrameType</u></a>	<a href="#"><u>OMX_VIDEO_VP8REFERENCEFRAMEINFOTYPE</u></a>
OMX_IndexParamVideoProfileLevelQuerySupported	OMX_VIDEO_PARAM_PROFILELEVELTYPE
OMX_IndexParamVideoProfileLevelCurrent	OMX_VIDEO_PARAM_PROFILELEVELTYPE
OMX_IndexConfigVideoBitrate	OMX_VIDEO_CONFIG_BITRATETYPE
OMX_IndexConfigVideoFramerate	OMX_CONFIG_FRAMERATETYPE
OMX_IndexConfigVideoIntraVOPRefresh	OMX_CONFIG_INTRAREFRESHVOPTYPE
OMX_IndexConfigVideoIntraMBRefresh	OMX_CONFIG_MACROBLOCKERRORMAPTYPE
OMX_IndexConfigVideoMBErrorReporting	OMX_CONFIG_MBERRORREPORTINGTYPE
OMX_IndexParamVideoMacroblocksPerFrame	OMX_PARAM_MACROBLOCKSTYPE
OMX_IndexConfigVideoMacroBlockErrorMap	OMX_CONFIG_MACROBLOCKERRORMAPTYPE
OMX_IndexParamVideoSliceFMO	OMX_VIDEO_PARAM_AVCSLICEFMO
OMX_IndexConfigVideoAVCIntraPeriod	OMX_VIDEO_CONFIG_AVCINTRAPERIOD
OMX_IndexConfigVideoNalSize	OMX_VIDEO_CONFIG_NALSIZE
<a href="#"><u>OMX_IndexParamNalStreamFormatSupported</u></a>	<a href="#"><u>OMX_NALSTREAMFORMATTYPE</u></a>
<a href="#"><u>OMX_IndexParamNalStreamFormat</u></a>	<a href="#"><u>OMX_NALSTREAMFORMATTYPE</u></a>
<a href="#"><u>OMX_IndexParamNalStreamFormatSelect</u></a>	<a href="#"><u>OMX_NALSTREAMFORMATTYPE</u></a>
<a href="#"><u>OMX_IndexParamVideoVC1</u></a>	<a href="#"><u>OMX_VIDEO_PARAM_VC1TYPE</u></a>
<a href="#"><u>OMX_IndexConfigVideoIntraPeriod</u></a>	<a href="#"><u>OMX_VIDEO_INTRAPERIODTYPE</u></a>

#### 4.3.3 ~~Video Use Case Examples~~

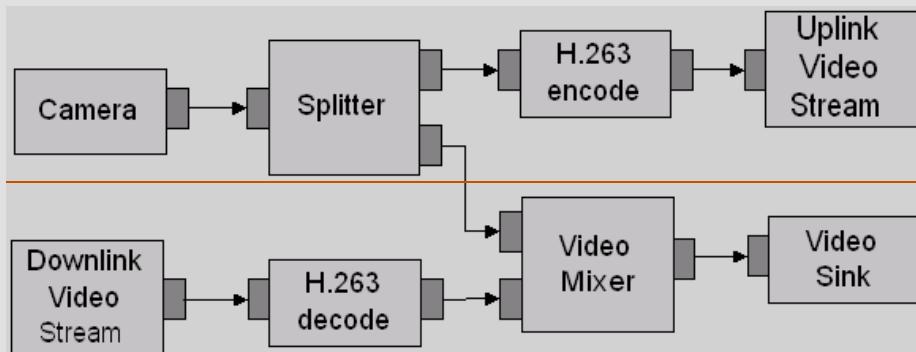
~~Figure 4-8 depicts one possible set of components as well as the tunneling of ports for these components to implement a H.263 video encoding scheme. This use case encodes raw video into H.263 format and writes it to a file while previewing the captured video on a display.~~



**Figure 4-8. H.263 Video Encode Use Case**

Figure 4-8 shows six components, namely the camera, the image filter, the splitter, the H.263 video encoder, the file writer, and the video sink.

Figure 4-9 shows a more complex use case, which is video conferencing. This use case supports simultaneous encoding and decoding of video streams. To simplify the use case, the corresponding audio components are not included.



**Figure 4-9. Video Conferencing Use Case**

Raw video is encoded to H.263 format and then transmitted via a video uplink to the far side conferencing participant. At the same time, a H.263 video stream is received from the far side participant via a video downlink and decoded to raw video format before being mixed into a pre-determined presentation layout via the video mixer such that both the local participant's video and far side participant's video are displayed via the local video sink.

#### 4.3.4 OMX\_VIDEO\_PORTDEFINITIONTYPE

The PortDefinition structure defines all of the parameters necessary for the compliant component to set up an input or an output video path. If additional information is needed to define the parameters of the port such as frame rate and bit rate, additional structures shall be sent. For example, to change the bit rate, send the OMX\_VIDEO\_PARAM\_BITRATETYPE structure to supply the extra parameters for the

port. The number of video paths for input and output will vary by the type of the video component.

The OMX\_VIDEO\_PORTDEFINITIONTYPE structure can query the current or default definition of a video port or set the definition of a video port for a component. The OMX\_VIDEO\_PORTDEFINITIONTYPE structure is included as part of the OMX\_PARAM\_PORTDEFINITIONTYPE structure, it is accessed via the OMX\_GetParameter function or the OMX\_SetParameter function using the OMX\_IndexParamPortDefinition index.

OMX\_VIDEO\_PORTDEFINITIONTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PORTDEFINITIONTYPE {
    OMX_STRING eMIMETYPE;
    OMX_NATIVE_DEVICETYPE pNativeRender;
    OMX_U32 nFrameWidth;
    OMX_U32 nFrameHeight;
    OMX_S32 nStride;
    OMX_S32 nSliceHeight;
    OMX_U32 nBitrate;
    OMX_U32 xFramerate;
    OMX_BOOL bFlagErrorConcealment;
    OMX_VIDEO_CODINGTYPE eCompressionFormat;
    OMX_COLOR_FORMATTYPE eColorFormat;
    OMX_NATIVE_WINDOWTYPE pNativeWindow;
} OMX_VIDEO_PORTDEFINITIONTYPE;
```

#### 4.3.4.1 Parameters

The parameters for OMX\_VIDEO\_PORTDEFINITIONTYPE are defined as follows.

- ~~eMIMETYPE is the MIME type of data for the port. If a MIME type string buffer is not supplied this parameter shall be set to NULL.~~
- ~~pNativeRender~~ is a platform specific reference for a render object. When the port is on a display sink component, this field is interpreted as a platform specific native display object when non-NUL. If NULL, the component uses the ~~pNativeWindow~~ field.
- ~~nFrameWidth~~ is the width of the data in pixels. If the value is 0x0 for an input port, the component will automatically detect and configure the width. For output ports, the width will be detected during ~~OMX\_SetupTunnel~~.
- ~~nFrameHeight~~ is the height of the data in pixels. If the value is 0x0 for an input port, the component will automatically detect and configure the height. For output ports, the height will be detected during ~~OMX\_SetupTunnel~~.
- ~~nStride~~ is a read-write field indicating the number of bytes per span of an image, where ~~nStride~~ is the amount added to go from span N to span N+1. A negative value for ~~nStride~~ indicates that the data is stored bottom-to-top instead of top-to-bottom. If the value is set to 0, the component automatically computes the value  
The value for nStride shall not be 0x0.

The nStride default shall be determined by the component. There are cases however when the default value for nStride does not match the stride requirements of a used buffer, or that of a tunneled port. [The IL client is allowed to overwrite this default value.](#)

~~Components shall validate the stride parameter when the port is enabled, or when the component is commanded from OMX\_StateLoaded to OMX\_StateIdle. The component may fail the transition if the specified stride is not supported.~~

- nSliceHeight is a read-writeonly field containing the slice height parameter used when processing uncompressed image data. Buffers received on the port shall contain integer multiples of slices. For more information on the minimum buffer payload for uncompressed data, see section 4.2.2. If the value is set to 0, the component automatically computes the value.  
  
The nSliceHeight default shall be determined by the component. There are cases however when the default value for nSliceHeight does not match the stride requirements of a used buffer, or that of a tunneled port. The IL client is allowed to overwrite this default value.
- nBitrate is the bit rate in bits per second of the frame to be used on the port if the data is compressed. The value 0x0 is used if the bit rate is unknown, variable or is not needed.
- xFramerate is the frame rate is in frames per second. This value is represented in Q16 format. ~~The frame rate specified is that used on the port if the data is not compressed.~~ The value 0x0 is used to indicate the frame rate is unknown, variable, or is not needed.
- bFlagErrorConcealment is a Boolean value that enables or disables error concealment if it is supported by the port.
- eCompressionFormat is the compression format used on the port. If the coding is being used to specify the ENCODE type, then additional work shall be done to configure the exact flavor of the compression to be used. For decode cases where the user application cannot differentiate between MPEG-4 and H.264 bit streams, the codec is responsible for the compression format. When OMX\_VIDEO\_CodingUnused is specified, the eColorFormat field is valid. For possible coding types, see Table 4-51.
- eColorFormat is the color format of the data for the port. This field is invalid unless the eCompressionFormat is OMX\_VIDEO\_CodingUnused. For more information on color format types, see Table 4-35.
- pNativeWindow is a platform specific reference for a windows object when being processed as part of a video sink component, otherwise this field is 0.

#### **4.3.4.2 Functionality**

When the IL client sets the `nFrameWidth` and `nFrameHeight` of the port for the first time, it can provide a value of 0 for `nStride` and `nSliceHeight`. Upon receiving the `OMX_SetParameter` call, the component computes updated values based on the settings of `nFrameWidth` and `nFrameHeight`. The IL client may retrieve the updated values via `OMX_GetParameter`.

If the IL client wishes to update and override these values, it may do so via `OMX_SetParameter` provided that the new value(s) are not less than the newly updated values. If the new values cannot be accommodated by the component, the component shall return `OMX_ErrorBadParameter`. This allows the OMX IL client to immediately be informed of the incompatibility.

By setting first a non-zero value for `nStride` (resp. `nSliceHeight`) and a zero value for `nSliceHeight` (resp. `nStride`), the IL client is able to benefit from the computation of another default value for `nSliceHeight` (resp. `nStride`). This may be relevant in cases when an override of `nStride` changes the default `nSliceHeight`.

Components shall validate `nStride` and `nSliceHeight`:

- When it is commanded from `OMX_StateLoaded` to `OMX_StateIdle` or during a port enable request. An `OMX_EventError` event with `OMX_ErrorPortsNotCompatible` shall be emitted if the port validation fails.
- During a tunnel setup call. `OMX_SetupTunnel` shall return `OMX_ErrorPortsNotCompatible` if the port validation fails.

#### **4.3.5 OMX\_VIDEO\_PARAM\_PORTFORMATTYPE**

`OMX_VIDEO_PARAM_PORTFORMATTYPE` is the structure for the port format parameter. It enumerates the various data input/output formats supported by the port.

`OMX_VIDEO_PARAM_PORTFORMATTYPE` can be used with both `OMX_GetParameter` and `OMX_SetParameter`. In the `OMX_GetParameter` case, the caller specifies all fields and the `OMX_GetParameter` call returns the value of `eFormat`. The value of `nIndex` is the range 0 to N-1, where N is the number of formats supported by the port. There is no need for the port to report N, as the caller can determine N by enumerating all the formats supported by the port. Each port shall support at least one format. If there are no more formats, `OMX_GetParameter` returns `OMX_ErrorNoMore` (i.e., `nIndex` is supplied where the value is N or greater). Ports supply formats in order of preference, which means that higher preference formats are provided with lower values of `nIndex`.

On `OMX_SetParameter`, the field in `nIndex` is ignored. If the format is supported, it is set as the format of the port, and the default values for the format are programmed

into the port definition type as a side effect. This allows the caller to query the default values for the format without having to know them in advance.

OMX\_VIDEO\_PARAM\_PORTFORMATTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_PORTFORMATTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nIndex;
    OMX_VIDEO_CODINGTYPE eCompressionFormat;
    OMX_COLOR_FORMATTYPE eColorFormat;
    OMX_U32 xFramerate;
} OMX_VIDEO_PARAM_PORTFORMATTYPE;
```

#### 4.3.5.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_PORTFORMATTYPE are defined as follows.

- nPortIndex is the read only value containing the index of the portrepresents the port that this structure applies to.
- nIndex indicates the enumeration index for the format from 0x0 to N-1.
- eCompressionFormat is the compression format used on the port. If the coding is being used to specify the ENCODE type, then additional work shall be done to configure the exact flavor of the compression to be used. For decode cases where the user application cannot differentiate between MPEG-4 and H.264 bit streams, the codec is responsible for the compression format. When OMX\_VIDEO\_CodingUnused is specified, the eColorFormat field is valid. For possible coding types, see Table 4-51.
- eColorFormat is the color format of the data for the port. This field is invalid unless the eCompressionFormat is OMX\_VIDEO\_CodingUnused. For more information on color format types, see Table 4-32: Uncompressed Data Formats
- xFramerate indicates the desired full frame rate is frames per second. This value is represented in Q16 format

#### 4.3.6 OMX\_VIDEO\_PARAM\_QUANTIZATIONTYPE

Quantization controls the compression used during the discrete cosine transform (DCT) step of video encoding. This generic structure is shared between several video standards. The structure allows independent settings of quantization factors for I, P, and B video frames. The structure is not applicable to variable bit rate encoding or constant rate encoding. Not all video standards support independent settings of quantization factors for different frame types.

OMX\_VIDEO\_PARAM\_QUANTIZATIONTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_QUANTIZATIONTYPE {
    OMX_U32 nSize;
```

```

OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_U32 nQpI;
OMX_U32 nQpP;
OMX_U32 nQpB;
} OMX_VIDEO_PARAM_QUANTIZATIONTYPE;

```

#### 4.3.6.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_QUANTIZATIONTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nQpI is the quantization parameter for I frames.
- nQpP is the quantization parameter for P frames.
- nQpB is the quantization parameter for bi-directional (B) frames.

#### 4.3.6.2 Dependencies

This parameter is only applicable to certain video encoders, which include MPEG-2 and MPEG-4.

### 4.3.7 OMX\_VIDEO\_PARAM\_VIDEOFASTUPDATETYPE

Video fast update is a shared parameter between multiple video encoding standards (for example, H.261 and H.263) that specifies fast update parameters for the video encoder.

OMX\_VIDEO\_PARAM\_VIDEOFASTUPDATETYPE is defined as follows.

```

typedef struct OMX_VIDEO_PARAM_VIDEOFASTUPDATETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnableVFU;
    OMX_U32 nFirstGOB;
    OMX_U32 nFirstMB;
    OMX_U32 nNumMBs;
} OMX_VIDEO_PARAM_VIDEOFASTUPDATETYPE;

```

#### 4.3.7.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_VIDEOFASTUPDATETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- bEnableVFU is a Boolean value that enables or disables video fast update.
- nFirstGOB contains the number of the first row of macroblocks

- nFirstMB is the location of the first macroblock row relative to the first group of blocks (GOB).
- nNumMBs The number of macroblocks to be refreshed from the nFirstGOB and nFirstMB.

#### 4.3.7.2 Dependencies

This parameter is only applicable to certain video encoders, such as H.261 and H.263.

#### 4.3.8 OMX\_VIDEO\_PARAM\_BITRATETYPE

Video encode bit rate control for variable bit rate video encoders is shared between multiple video encode standards, and is specified before starting video encoding.

OMX\_VIDEO\_PARAM\_BITRATETYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_BITRATETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_VIDEO_CONTROLRATETYPE eControlRate;
    OMX_U32 nTargetBitrate;
} OMX_VIDEO_PARAM_BITRATETYPE;
```

#### 4.3.8.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_BITRATETYPE are defined as follows.

- nPortIndex *is the read-only value containing the index of the portrepresents the port that this structure applies to.*
- eControlRate is an enumerated value that sets the bit rate control. If enabled, the type of bit rate control is specified as constant, variable, constant with frame skipping, or variable with frame skipping. Table 4-54 enumerates the possible video bit rate control types for OMX\_VIDEO\_CONTROLRATETYPE.

**Table 4-54: Supported Video Bit Rate Control Types**

Field Name	Bit Rate Control Descriptions
OMX_Video_ControlRateDisable	Disable – in this mode the encoder will ignore nTargetBitrate setting and use the appropriate Qp (nQpI, nQpP, nQpB) values for encoding
OMX_Video_ControlRateVariable	Variable bit rate
OMX_Video_ControlRateConstant	Constant bit rate – the encoder can modify the Qp values to meet the nTargetBitrate target
OMX_Video_ControlRateVariableSkipFrames	Variable bit rate with frame skipping

Field Name	Bit Rate Control Descriptions
OMX_Video_ControlRateConstantSkipFrames	Constant bit rate with frame skipping – the encoder cannot modify the Qp values to meet the nTargetBitrate target. Instead, the encoder can drop frames to achieve nTargetBitrate
OMX_Video_ControlRateMax	Maximum value

- nTargetBitrate is the target bit rate for video encoding in units of bits per second.

#### 4.3.8.2 Dependencies

This parameter is only applicable to certain video encoders. For some video encode standards, the bit rate is specified as part of the standard and is not programmable (i.e., value can only be queried).

### 4.3.9 OMX\_VIDEO\_PARAM\_MOTIONVECTORTYPE

The motion vector parameters used during video encoding are programmable for certain video standards. These parameters can be shared between multiple video standards algorithms, although certain fields only pertain to particular video standards.

OMX\_VIDEO\_PARAM\_MOTIONVECTORTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_MOTIONVECTORTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_VIDEO_MOTIONVECTORTYPE eAccuracy;
    OMX_BOOL bUnrestrictedMVs;
    OMX_BOOL bFourMV;
    OMX_S32 sXSearchRange;
    OMX_S32 sYSearchRange;
} OMX_VIDEO_PARAM_MOTIONVECTORTYPE;
```

#### 4.3.9.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_MOTIONVECTORTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- eAccuracy is an enumerated value that specifies the pixel accuracy of the motion vector search during video encode. Accuracy is 1, 1/2, 1/4, or 1/8 pixel. The eAccuracy setting indicates that all larger value motion vector search ranges are also used (i.e., a value of 1/4 indicates motion vectors are also searched

on 1 and 1/2 intervals). Table 4-55 enumerates the possible video motion vector types for OMX\_VIDEO\_MOTIONVECTORTYPE.

**Table 4-55: Supported Video Motion Vector Types**

Field Name	Motion Vector Descriptions
OMX_Video_MotionVectorPixel	Full pixel motion vectors
OMX_Video_MotionVectorHalfPel	Half pixel motion vectors
OMX_Video_MotionVectorQuarterPel	Quarter pixel motion vectors
OMX_Video_MotionVectorEighthPel	Eighth pixel motion vectors
<b>OMX_Video_MotionVectorMax</b>	<b>Maximum value</b>

- bUnrestrictedMVs is a Boolean value that enables unrestricted motion vectors.
- bFourMV is a Boolean value enables using four motion vectors.
- sXSearchRange is the search range of the X motion vector in pixels for video encoders where this is programmable. For example, a search range of 4 indicates a  $\pm 4$  search area both horizontally and vertically.
- sYSearchRange is the search range of the Y motion vector in pixels for video encoders where this is programmable. For example, a search range of 4 indicates a  $\pm 4$  search area both horizontally and vertically.

#### 4.3.9.2 Dependencies

This parameter is only applicable to certain video encoders, which include MPEG2 and MPEG4.

#### 4.3.10 OMX\_VIDEO\_PARAM\_INTRAREFRESHTYPE

OMX\_VIDEO\_PARAM\_INTRAREFRESHTYPE contains common parameters for controlling the intra-refresh rate for macroblocks during video encoding. Refresh causes macroblocks of a video stream to be regularly encoded as reference macroblocks. This enables a video decoder to eventually reconstruct a good video image from multiple frames when data is lost or corrupted without receiving a new intra-coded frame.

OMX\_VIDEO\_PARAM\_INTRAREFRESHTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_INTRAREFRESHTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_VIDEO_INTRAREFRESHTYPE eRefreshMode;
    OMX_U32 nAirMBs;
    OMX_U32 nAirRef;
    OMX_U32 nCirMBs;
} OMX_VIDEO_PARAM_INTRAREFRESHTYPE;
```

#### 4.3.10.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_INTRAREFRESHTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- eRefreshMode is the enumeration for the type of intra-refresh mode. Table 4-56 shows the possible values for OMX\_VIDEO\_INTRAREFRESHTYPE .

Table 4-56: Supported Video Intra-Refresh Types

Field Name	Intra-Refresh Descriptions
OMX_VIDEO_IntraRefreshCyclic	Cyclic intra-refresh
OMX_VIDEO_IntraRefreshAdaptive	Adaptive intra-refresh
OMX_VIDEO_IntraRefreshBoth	Cyclic and Adaptive intra-refresh
OMX_VIDEO_IntraRefreshMax	Maximum value

- nAirMBs is the minimum number of macroblocks to refresh in a frame when adaptive intra-refresh (AIR) is enabled.
- nAirRef is the number of times a motion marked macroblock has to be intra-coded.
- nCirMBs is the number of consecutive macroblocks to be coded as intra when cyclic intra-refresh (CIR) is enabled.

#### 4.3.10.2 Dependencies

This parameter is only applicable to certain video encoders, which includes MPEG4.

#### 4.3.11 OMX\_VIDEO\_PARAM\_ERRORCORRECTIONTYPE

OMX\_VIDEO\_PARAM\_ERRORCORRECTIONTYPE contains common video encoding standard parameters for handling error correction during video encoding.

OMX\_VIDEO\_PARAM\_ERRORCORRECTIONTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_ERRORCORRECTIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnableHEC;
    OMX_BOOL bEnableResync;
    OMX_U32 nResynchMarkerSpacing;
    OMX_BOOL bEnableDataPartitioning;
    OMX_BOOL bEnableRVLC;
} OMX_VIDEO_PARAM_ERRORCORRECTIONTYPE;
```

#### 4.3.11.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_ERRORCORRECTIONTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- bEnableHEC is a Boolean value that enables or disables header extension codes.
- bEnableResync is a Boolean value that enables or disables resynchronization markers.
- nResynchMarkerSpacing is the resynchronization marker interval in bits applied to the stream.
- bEnableDataPartitioning is a Boolean value that enables or disables data partitioning.
- bEnableRVLC is a Boolean value that enables or disables reversible variable-length coding.

#### 4.3.11.2 Dependencies

This parameter is only applicable to certain video encoders, which includes MPEG4.

### 4.3.12 OMX\_VIDEO\_PARAM\_VBSMCTYPE

OMX\_VIDEO\_PARAM\_VBSMCTYPE contains common video encoding standard parameters for selecting variable block size motion compensation during video encoding.

OMX\_VIDEO\_PARAM\_VBSMCTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_VBSMCTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL b16x16;
    OMX_BOOL b16x8;
    OMX_BOOL b8x16;
    OMX_BOOL b8x8;
    OMX_BOOL b8x4;
    OMX_BOOL b4x8;
    OMX_BOOL b4x4;
} OMX_VIDEO_PARAM_VBSMCTYPE;
```

#### 4.3.12.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_VBSMCTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.

- b16x16 is a Boolean value that enables or disables inter-block search in a 16 by 16 region of pixels
- b16x8 is a Boolean value that enables or disables inter-block search in a 16 by 8 region of pixels
- b8x16 is a Boolean value that enables or disables inter-block search in a 8 by 16 region of pixels
- b8x8 is a Boolean value that enables or disables inter-block search in a 8 by 8 region of pixels
- b8x4 is a Boolean value that enables or disables inter-block search in a 8 by 4 region of pixels
- b4x8 is a Boolean value that enables or disables inter-block search in a 4 by 8 region of pixels
- b4x4 is a Boolean value that enables or disables inter-block search in a 4 by 4 region of pixels

#### **4.3.12.2 Dependencies**

This parameter is only applicable to certain video encoders, which include MPEG4 and other derivations of MPEG4.

#### **4.3.13 OMX\_VIDEO\_PARAM\_H263TYPE**

H.263 is a video standard defined by the ITU. Parameters for this video standard are controlled using the OMX\_VIDEO\_PARAM\_H263TYPE structure.

OMX\_VIDEO\_PARAM\_H263TYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_H263TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nPFrames;
    OMX_U32 nBFrames;
    OMX_VIDEO_H263PROFILETYPE eProfile;
    OMX_VIDEO_H263LEVELTYPE eLevel;
    OMX_BOOL bPLUSPTYPEAllowed;
    OMX_U32 nAllowedPictureTypes;
    OMX_BOOL bForceRoundingTypeToZero;
    OMX_U32 nPictureHeaderRepetition;
    OMX_U32 nGOBHeaderInterval;
} OMX_VIDEO_PARAM_H263TYPE;
```

##### **4.3.13.1 Parameters**

The parameters for OMX\_VIDEO\_PARAM\_H263TYPE are defined as follows.

- nPortIndex is the read only value containing the index of the portrepresents the port that this structure applies to.
- nPFrames is the number of P frames between I frames.
- nBFrames is the number of B frames between I frames.
- eProfile is the profile type supported for encoding and decoding H.263 content. Table 4-57 shows the possible H.263 video profile types for OMX\_VIDEO\_H263PROFILETYPE.

**Table 4-57: Supported H.263 Profile Types**

Field Name	H.263 Profile Descriptions
OMX_VIDEO_H263ProfileBaseline	<u>Unknown, unused or not required profile setting</u> <u>H.263 Baseline Profile: H.263 (V1), no optional modes,</u>
<u>OMX_VIDEO_H263ProfileBaseline</u>	<u>H.263 Baseline Profile: H.263 (V1), no optional modes</u>
OMX_VIDEO_H263ProfileH320Coding	H.263 Coding Efficiency (H.320) Backward Compatibility Profile: H.263+ (V2), includes annexes I, J, L.4, and T
OMX_VIDEO_H263ProfileBackward Compatible	H.263 BackwardCompatible: Backward Compatibility Profile: H.263 (V1), includes annex F
OMX_VIDEO_H263ProfileISWV2	H.263 Interactive Streaming Wireless Profile: H.263+ (V2), includes annexes I, J, K, and T
OMX_VIDEO_H263ProfileISWV3	H.263 Interactive Streaming Wireless Profile: H.263++ (V3), includes profile 3 and annexes V and W.6.3.8
OMX_VIDEO_H263ProfileHigh Compression	H.263 Conversational High Compression Profile: H.263++ (V3), includes profiles 1 and 2 and annexes D and U
OMX_VIDEO_H263ProfileInternet	H.263 Conversational Internet Profile: H.263++ (V3), includes profile 5 and annex K
OMX_VIDEO_H263ProfileInterlace	H.263 Conversational Interlace Profile: H.263++ (V3), includes profile 5 and annex W.6.3.11
OMX_VIDEO_H263ProfileHighLatency	H.263 High Latency Profile: H.263++ (V3), includes profile 6 and annexes O.1 and P.5
<u>OMX_VIDEO_H263ProfileMax</u>	<u>Maximum value</u>

- eLevel is the maximum processing level that an encoder or decoder supports for a particular profile. Table 4-58 shows the possible H.263 video level types.

**Table 4-58: Supported H.263 Level Types**

Field Name	H.263 Level Descriptions
<a href="#">OMX_VIDEO_H263Level10</a>	<a href="#">H.263 level 10</a>
<a href="#">OMX_VIDEO_H263LevelUnknown</a>	<a href="#">Unknown, unused or not required setting.</a>
<a href="#">OMX_VIDEO_H263Level10</a>	<a href="#">H.263 level 10</a>
OMX_VIDEO_H263Level120	H.263 level 20
OMX_VIDEO_H263Level130	H.263 level 30
OMX_VIDEO_H263Level140	H.263 level 40
OMX_VIDEO_H263Level145	H.263 level 45
OMX_VIDEO_H263Level150	H.263 level 50
OMX_VIDEO_H263Level160	H.263 level 60
OMX_VIDEO_H263Level170	H.263 level 70
<a href="#">OMX_VIDEO_H263LevelMax</a>	<a href="#">Maximum value</a>

- bPLUSPTYPEAllowed is a Boolean value that enables or disables indication of whether PLUSPTYPE (specified in the 1998 version of H.263) is allowed. This applies to custom picture sizes or clock frequencies.
- nAllowedPictureTypes determines whether picture types are allowed in the bit stream. For more information on picture types, see Table 4-52.
- bForceRoundingTypeToZero determines whether the value of the RTYPE bit (bit 6 of MPPTYPE) is not constrained. Change the value of the RTYPE bit for each reference picture in error-free communication.
- nPictureHeaderRepetition is the frequency of picture header repetition.
- nGOBHeaderInterval is the interval of non-empty GOB headers in units of GOBs. A value of zero for this parameter indicates that all GOB headers will be empty.

#### 4.3.13.2 Dependencies

This parameter is only applicable when the port is configured for H.263.

#### 4.3.14 OMX\_VIDEO\_PARAM\_MPEG2TYPE

OMX\_VIDEO\_PARAM\_MPEG2TYPE contains MPEG2 video parameters for controlling MPEG2 video encode.

OMX\_VIDEO\_PARAM\_MPEG2TYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_MPEG2TYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nPFrames;  
    OMX_U32 nBFrames;
```

```

    OMX_VIDEO_MPEG2PROFILETYPE eProfile;
    OMX_VIDEO_MPEG2LEVELTYPE eLevel;
} OMX_VIDEO_PARAM_MPEG2TYPE;

```

#### 4.3.14.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_MPEG2TYPE are defined as follows.

- nPortIndex *is the read-only value containing the index of the port that represents the port that this structure applies to.*
- nPFrames is the number of P frames between I frames.
- nBFrames is the number of B frames between I frames.
- eProfile is the maximum processing level that an encoder or decoder supports for a particular profile. Table 4-59 shows the possible MPEG-2 video profile types in OMX\_VIDEO\_MPEG2PROFILETYPE.

**Table 4-59: Supported MPEG-2 Profile Types**

Field Name	MPEG-2 Profile Descriptions
<a href="#">OMX_VIDEO_MPEG2ProfileSimple</a>	<a href="#">Simple profile</a>
<a href="#">OMX_VIDEO_MPEG2ProfileUnknown</a>	<a href="#">Unknown, unused or not required profile setting.</a>
<a href="#">OMX_VIDEO_MPEG2ProfileSimple</a>	<a href="#">Simple profile</a>
OMX_VIDEO_MPEG2ProfileMain	Main profile
OMX_VIDEO_MPEG2Profile422	4:2:2 profile
OMX_VIDEO_MPEG2ProfileSNR	SNR profile
OMX_VIDEO_MPEG2ProfileSpatial	Spatial profile
OMX_VIDEO_MPEG2ProfileHigh	High profile
<a href="#">OMX_VIDEO_MPEG2ProfileMax</a>	<a href="#">Maximum value</a>

- eLevel is the maximum processing level that an MPEG-2 encoder or decoder supports for a particular profile. Table 4-60 shows the possible MPEG-2 video level types in OMX\_VIDEO\_MPEG2LEVELTYPE.

**Table 4-60: Supported MPEG-2 Level Types**

Field Name	MPEG-2 Level Descriptions
<a href="#">OMX_VIDEO_MPEG2LevelH</a>	<a href="#">Low level</a>
<a href="#">OMX_VIDEO_MPEG2LevelUnknown</a>	<a href="#">Unknown, unused or not required setting.</a>
<a href="#">OMX_VIDEO_MPEG2LevelLL</a>	<a href="#">Low level</a>
OMX_VIDEO_MPEG2LevelML	Main level
OMX_VIDEO_MPEG2LevelH14	High 1440 level
OMX_VIDEO_MPEG2LevelHL	High level
<a href="#">OMX_VIDEO_MPEG2LevelMax</a>	<a href="#">Maximum level</a>

#### 4.3.14.2 Dependencies

This parameter is only applicable when the port is configured for MPEG-2.

#### 4.3.15 OMX\_VIDEO\_PARAM\_MPEG4TYPE

OMX\_VIDEO\_PARAM\_MPEG4TYPE contains the MPEG-4 video parameters for controlling MPEG-4 video encoding and decoding.

OMX\_VIDEO\_PARAM\_MPEG4TYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_MPEG4TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nSliceHeaderSpacing;
    OMX_BOOL bSVH;
    OMX_BOOL bGov;
    OMX_U32 nPFrames;
    OMX_U32 nBFrames;
    OMX_U32 nIDCVLCThreshold;
    OMX_BOOL bACPred;
    OMX_U32 nMaxPacketSize;
    OMX_U32 nTimeIncRes;
    OMX_VIDEO_MPEG4PROFILETYPE eProfile;
    OMX_VIDEO_MPEG4LEVELTYPE eLevel;
    OMX_U32 nAllowedPictureTypes;
    OMX_U32 nHeaderExtension;
    OMX_BOOL bReversibleVLC;
} OMX_VIDEO_PARAM_MPEG4TYPE;
```

#### 4.3.15.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_MPEG4TYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nSliceHeaderSpacing is the number of macroblocks in a slice (H263+ Annex K). This value shall be zero if not used.

- bSVH is a Boolean value that enables or disables short header mode.
- bGov is a Boolean value that enables or disables group of VOP (GOV), where VOP is the abbreviation for video object planes.
- nPFrames is the number of P frames between I frames.
- nBFrames is the number of B frames between I frames.
- nIDCVLCThreshold is the value of the intra-DC variable-length coding (VLC) threshold.
- bACPred is the Boolean value that enables or disables AC prediction.
- nMaxPacketSize is the maximum size of the packet in bytes.
- nTimeIncRes is the VOP time increment resolution for MPEG-4. This value is interpreted as described in the MPEG-4 standard.
- eProfile is the profile used for MPEG-4 encoding or decoding. Table 4-61 shows the possible MPEG-4 video profile types in OMX\_VIDEO\_MPEG4PROFILETYPE.

**Table 4-61: Supported MPEG-4 Profile Types**

Field Name	MPEG-4 Profile Descriptions
<a href="#">OMX_VIDEO_MPEG4ProfileSimple</a>	MPEG-4 Simple Profile, Levels 1-3
<a href="#">OMX_VIDEO_MPEG4ProfileUnknown</a>	Unknown, unused or not required profile setting.
<a href="#">OMX_VIDEO_MPEG4ProfileSimple</a>	MPEG-4 Simple Profile, Levels 1-3
OMX_VIDEO_MPEG4ProfileSimpleScalable	MPEG-4 Simple Scalable Profile, Levels 1-2
OMX_VIDEO_MPEG4ProfileCore	MPEG-4 Core Profile, Levels 1-2
OMX_VIDEO_MPEG4ProfileMain	MPEG-4 Main Profile, Levels 2-4
OMX_VIDEO_MPEG4ProfileNbit	MPEG-4 N-bit Profile, Level 2
OMX_VIDEO_MPEG4ProfileScalableTexture	MPEG-4 Scalable Texture Profile, Level 1
OMX_VIDEO_MPEG4ProfileSimpleFace	MPEG-4 Simple Face Animation Profile, Levels 1-2
OMX_VIDEO_MPEG4ProfileSimpleFBA	MPEG-4 Simple Face and Body Animation (FBA) Profile, , Levels 1-2
OMX_VIDEO_MPEG4ProfileBasicAnimated	MPEG-4 Basic Animated Texture Profile, Levels 1-2
OMX_VIDEO_MPEG4ProfileHybrid	MPEG-4 Hybrid Profile, Levels 1-2
OMX_VIDEO_MPEG4ProfileAdvancedRealTime	MPEG-4 Advanced Real Time Simple Profiles, Levels 1-4
OMX_VIDEO_MPEG4ProfileCoreScalable	MPEG-4 Core Scalable Profile, Levels 1-3

Field Name	MPEG-4 Profile Descriptions
OMX_VIDEO_MPEG4ProfileAdvancedCoding	MPEG-4 Advanced Coding Efficiency Profile, Levels 1-4
OMX_VIDEO_MPEG4ProfileAdvancedCore	MPEG-4 Advanced Core Profile, Levels 1-2
OMX_VIDEO_MPEG4ProfileAdvancedScalable	MPEG-4 Advanced Scalable Texture, Levels 2-3
OMX_VIDEO_MPEG4ProfileAdvancedSimple	MPEG-4 Advanced Simple Profile
<del>OMX_VIDEO_MPEG4ProfileMax</del>	<del>Maximum value</del>

- eLevel is the maximum processing level that an encoder or decoder supports for a particular MPEG-4 profile. Table 4-62 shows the possible MPEG-4 video level types in OMX\_VIDEO\_MPEG4LEVELTYPE.

Table 4-62: Supported MPEG-4 Level Types

Field Name	MPEG-4 Level Descriptions
<del>OMX_VIDEO_MPEG4Level0</del>	<del>Level0</del>
<del>OMX_VIDEO_MPEG4LevelUnknown</del>	<del>Unknown, unused or not required setting.</del>
<del>OMX_VIDEO_MPEG4Level0</del>	<del>Level 0</del>
OMX_VIDEO_MPEG4Level0b	Level 0b
OMX_VIDEO_MPEG4Level1	Level 1
OMX_VIDEO_MPEG4Level2	Level 2
OMX_VIDEO_MPEG4Level3	Level 3
OMX_VIDEO_MPEG4Level4	Level 4
OMX_VIDEO_MPEG4Level4a	Level 4a
OMX_VIDEO_MPEG4Level5	Level 5
<del>OMX_VIDEO_MPEG4LevelMax</del>	<del>Max level</del>

- nAllowedPictureTypes identifies the picture types allowed in the bit stream. For more information on picture types, see Table 4-52: Supported Video Picture Types.
- nHeaderExtension specifies the number of consecutive video packets between header extension codes (conversely, insert a header extension code every nHeaderExtension number of packets).
- bReversibleVLC is a Boolean value that enables or disables the use of reversible variable-length coding

#### 4.3.15.2 Dependencies

This parameter is only applicable when the port is configured for MPEG-4.

#### **4.3.16 OMX\_VIDEO\_PARAM\_WMVTYPE**

OMX\_VIDEO\_PARAM\_WMVTYPE contains common standard video decoder parameters that control Windows Media formats, including WMV7, WMV8, and WMV9.

OMX\_VIDEO\_PARAM\_WMVTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_WMVTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_VIDEO_WMVFORMATTYPE eFormat;
} OMX_VIDEO_PARAM_WMVTYPE;
```

##### **4.3.16.1 Parameters**

The parameters for OMX\_VIDEO\_PARAM\_WMVTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- eFormat is the enumerated format of the data stream. Table 4-63 shows the possible Windows Media video format types for OMX\_VIDEO\_WMVFORMATTYPE.

**Table 4-63: Supported Windows Media Video Format Types**

Field Name	Windows Media Video Format Descriptions
<u>OMX_VIDEO_WMVFormatUnused</u>	Format unused or unknown
<u>OMX_VIDEO_WMVFormatUnknown</u>	Unknown, unused or not required setting.
<u>OMX_VIDEO_WMVFormatUnused</u>	Format unused or unknown
OMX_VIDEO_WMVFormat7	Windows Media video format 7
OMX_VIDEO_WMVFormat8	Windows Media video format 8
OMX_VIDEO_WMVFormat9	Windows Media video format 9
<u>OMX_VIDEO_WMVFormatMax</u>	Maximum level

##### **4.3.16.2 Dependencies**

This parameter is only applicable when the port is configured for Windows Media video.

#### **4.3.17 OMX\_VIDEO\_PARAM\_RVTYPE**

OMX\_VIDEO\_PARAM\_RVTYPE contains common standard video decoder parameters that control RealVideo formats, including RealVideo 8 and RealVideo 9.

OMX\_VIDEO\_PARAM\_RVTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_RVTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_VIDEO_RVFFORMATTYPE eFormat;
    OMX_U16 nBitsPerPixel;
    OMX_U16 nPaddedWidth;
    OMX_U16 nPaddedHeight;
    OMX_U32 nFrameRate;
    OMX_U32 nBitstreamFlags;
    OMX_U32 nBitstreamVersion;
    OMX_U32 nMaxEncodeFrameSize;
    OMX_BOOL bEnablePostFilter;
    OMX_BOOL bEnableTemporalInterpolation;
    OMX_BOOL bEnableLatencyMode;
} OMX_VIDEO_PARAM_RVTYPE;
```

##### **4.3.17.1 Parameters**

The parameters for OMX\_VIDEO\_PARAM\_RVTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- eFormat is the video format. Table 4-64 shows the possible RealVideo video format types in OMX\_VIDEO\_RVFFORMATTYPE.

**Table 4-64: Supported RealVideo Format Types**

Field Name	RV Format Descriptions
<a href="#">OMX_VIDEO_RVFormatUnused</a>	Format unused or unknown
<a href="#">OMX_VIDEO_RVFormatUnknown</a>	Unknown, unused or not required setting
<a href="#">OMX_VIDEO_RVFormatUnused</a>	Format unused or unknown
OMX_VIDEO_RVFormat8	RealVideo 8 format
OMX_VIDEO_RVFormat9	RealVideo 9 format
OMX_VIDEO_RVFormatG2	RealVideo G2 format

- nBitsPerPixel is the number of bits per pixel coded in the frame.
- nPaddedWidth is the padded width in pixels of a video frame.
- nPaddedHeight is the padded height in pixels of a video frame.

- `nFrameRate` is the rate of the video in frames per second as a 32-bit fixed point value in which the upper 16 bits are the integer part and the lower 16 bits are the fractional part.
- `nBitstreamFlags` is a 32 bit integer containing flags which provide internal information about the bitstream to the codec. These will be interpreted differently depending on the bitstream format and version.
- `nBitstreamVersion` is a 32 bit integer containing the bitstream version.
- `nMaxEncodeFrameSize` is the size in bytes of the largest encoded frame (defined only for `OMX_VIDEO_RVFormat9`).
- `bEnablePostFilter` is a Boolean value that enables or disables the post filter.
- `bEnableTemporalInterpolation` a Boolean value that enables or disables the temporal interpolation.
- `bEnableLatencyMode` is a Boolean value that enables or disables the decoder from displaying a decoded frame until it has detected that no enhancement layer frames or dependent B frames will be coming. This detection usually occurs when a subsequent non-B frame is encountered.

#### **4.3.17.2 Dependencies**

This parameter is only applicable when the port is configured for RealVideo.

#### **4.3.18 OMX\_VIDEO\_PARAM\_AVCTYPE**

MPEG4 P10 Advanced Video Coding (AVC) is commonly referred to as H.264 which is a video standard defined by the Joint Video Team (JVT). Parameters for this video standard are controlled using the `OMX_VIDEO_PARAM_AVCTYPE` structure.

`OMX_VIDEO_PARAM_AVCTYPE` is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_AVCTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nSliceHeaderSpacing;
    OMX_U32 nPFrames;
    OMX_U32 nBFrames;
    OMX_BOOL bUseHadamard;
    OMX_U32 nRefFrames;
    OMX_U32 nRefIdx10ActiveMinus1;
    OMX_U32 nRefIdx11ActiveMinus1;
    OMX_BOOL bEnableUEP;
    OMX_BOOL bEnableFMO;
    OMX_BOOL bEnableASO;
    OMX_BOOL bEnableRS;
    OMX_VIDEO_AVCPROFILETYPE eProfile;
    OMX_VIDEO_AVCLEVELTYPE eLevel;
    OMX_U32 nAllowedPictureTypes;
    OMX_BOOL bFrameMBsOnly;
```

```

OMX_BOOL bMBAFF;
OMX_BOOL bEntropyCodingCABAC;
OMX_BOOL bWeightedPPrediction;
OMX_U32 nWeightedBipredictionMode;
OMX_BOOL bconstIpred ;
OMX_BOOL bDirect8x8Inference;
OMX_BOOL bDirectSpatialTemporal;
OMX_U32 nCabacInitIdc;
OMX_VIDEO_AVCLOOPFILERTYPE eLoopFilterMode;

} OMX_VIDEO_PARAM_AVCTYPE;

```

#### 4.3.18.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_AVCTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nSliceHeaderSpacing is the number of macroblocks in a slice. This value is set to 0x0 when not used.
- nPFrames is the number of P frames between I frames.
- nBFrames is the number of B frames between I frames.
- bUseHadamard is a Boolean value that enables or disables the Hadamard transform.
- nRefFrames is the number of reference frames in the range 1 to 16 that are used for inter-motion search.
- nRefIdx10ActiveMinus1 is the picture parameter set reference frame index, which is the index into the reference frame buffer of the trailing frames list. This value supports B frames.
- nRefIdx11ActiveMinus1 is the picture parameter set reference frame index, which is the index into the reference frame buffer of the forward frames list. This value supports B frames.
- bEnableUEP is a Boolean value that enables or disables unequal error protection. This parameter is only applicable if data partitioning is enabled.
- bEnableFMO is a Boolean value that enables or disables flexible macroblock ordering.
- bEnableASO is a Boolean value that enables or disables for arbitrary slice ordering.
- bEnableRS is a Boolean value enables or disables sending redundant slices.
- eProfile is the profile used for the types of AVC encoding or decoding that are supported. Table 4-65 shows the possible AVC video profile types in OMX\_VIDEO\_AVCPROFILETYPE.

**Table 4-65: Supported AVC Profile Types**

Field Name	AVC Profile Descriptions
<code>OMX_VIDEO_AVCPProfileBaseline</code>	<u>Baseline profile</u>
<code>OMX_VIDEO_AVCPProfileUnknown</code>	<u>Unknown, unused or not required profile setting.</u>
<code>OMX_VIDEO_AVCPProfileBaseline</code>	<u>Baseline profile</u>
<code>OMX_VIDEO_AVCPProfileMain</code>	Main profile
<code>OMX_VIDEO_AVCPProfileExtended</code>	Extended profile
<code>OMX_VIDEO_AVCPProfileHigh</code>	High profile
<code>OMX_VIDEO_AVCPProfileHigh10</code>	High 10 profile
<code>OMX_VIDEO_AVCPProfileHigh422</code>	High 4:2:2 profile
<code>OMX_VIDEO_AVCPProfileHigh444</code>	High 4:4:4 profile
<code>OMX_VIDEO_AVCPProfileMax</code>	<u>Maximum value</u>

- eLevel is the maximum processing level that an AVC encoder or decoder supports for a particular profile. Table 4-66 shows the possible AVC video level types in `OMX_VIDEO_AVCLEVELTYPE`.

**Table 4-66: Supported AVC Level Types**

Field Name	AVC Level Descriptions
<code>OMX_VIDEO_AVCLevel1</code>	<u>AVC level 1</u>
<code>OMX_VIDEO_AVCLevelUnknown</code>	<u>Unknown, unused or not required setting.</u>
<code>OMX_VIDEO_AVCLevel1</code>	<u>AVC level 1</u>
<code>OMX_VIDEO_AVCLevel1b</code>	AVC level 1b
<code>OMX_VIDEO_AVCLevel11</code>	AVC level 1.1
<code>OMX_VIDEO_AVCLevel12</code>	AVC level 1.2
<code>OMX_VIDEO_AVCLevel13</code>	AVC level 1.3
<code>OMX_VIDEO_AVCLevel2</code>	AVC level 2
<code>OMX_VIDEO_AVCLevel21</code>	AVC level 2.1
<code>OMX_VIDEO_AVCLevel22</code>	AVC level 2.2
<code>OMX_VIDEO_AVCLevel3</code>	AVC level 3
<code>OMX_VIDEO_AVCLevel31</code>	AVC level 3.1
<code>OMX_VIDEO_AVCLevel32</code>	AVC level 3.2
<code>OMX_VIDEO_AVCLevel4</code>	AVC level 4
<code>OMX_VIDEO_AVCLevel41</code>	AVC level 14.1
<code>OMX_VIDEO_AVCLevel42</code>	AVC level 4.2
<code>OMX_VIDEO_AVCLevel5</code>	AVC level 5
<code>OMX_VIDEO_AVCLevel51</code>	AVC level 5.1
<code>OMX_VIDEO_AVCLevelMax</code>	<u>Maximum value</u>

- nAllowedPictureTypes identifies the allowed picture types in the bit stream.
- bFrameMBsOnly is a Boolean value indicating that every coded picture of the coded video sequence is a coded frame containing only frame macroblocks.
- bMBAFF is a Boolean value that enables or disables macroblock adaptive frame and field (MBAFF) support within a picture.
- bEntropyCodingCABAC is a Boolean value that enables or disables the entropy decoding method.
- bWeightedPPrediction is a Boolean value that enables or disables weighted prediction applied to P and SP slices.
- nWeightedBipredictionMode is the default weighted prediction applied to B slices.
- bconstIpred is a Boolean value that enables or disables intra-prediction.
- bDirect8x8Inference specifies the method used in the derivation process for luma motion vectors for B\_Skip, B\_Direct\_16x16, and B\_Direct\_8x8 as specified in subclause 8.4.1.2 of the AVC spec.
- bDirectSpatialTemporal is a flag that indicates the spatial or temporal direct mode used in B-slice coding, which is related to bDirect8x8Inference . Spatial direct mode is the default.
- nCabacInitIdx is the index used to initialize Context-based Adaptive Binary Arithmetic Coding (CABAC) contexts.
- eLoopFilterMode enables or disables the AVC loop filter. Table 4-67 shows the possible AVC video coding loop filter types in OMX\_VIDEO\_AVCLOOPFILTERTYPE.

**Table 4-67: Supported AVC Loop Filter Types**

Field Name	AVC Loop Filter Level Descriptions
OMX_VIDEO_AVCLoopFilterEnable	Enables AVC loop filter
OMX_VIDEO_AVCLoopFilterDisable	Disables AVC loop filter
OMX_VIDEO_AVCLoopFilterDisableSliceBoundary	Disables AVC loop filter on slice boundary
OMX_VIDEO_AVCLevelMax	Maximum level

#### 4.3.18.2 Dependencies

This parameter is only applicable when the port is configured for AVC.

#### 4.3.19 OMX VIDEO PARAM VP8TYPE

OMX\_VIDEO\_PARAM\_VP8TYPE contains the VP8 video parameters for controlling VP8 video encoding and decoding.

OMX\_VIDEO\_PARAM\_VP8TYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_VP8TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_VIDEO_VP8PROFILETYPE eProfile;
    OMX_VIDEO_VP8LEVELTYPE eLevel;
    OMX_U32 nDCTPartitions;
    OMX_BOOL bErrorResilientMode;
} OMX_VIDEO_PARAM_VP8TYPE;
```

#### 4.3.19.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_VP8TYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- eProfile is the profile used for the types of VP8 encoding or decoding that are supported.

Table 4-68 shows the possible VP8 video profile types in OMX\_VIDEO\_VP8PROFILETYPE.

Table 4-68: Supported VP8 Profile Types

<u>Field Name</u>	<u>VP8 Profile Descriptions</u>
<u>OMX_VIDEO_VP8ProfileUnknown</u>	<u>Unknown, unused or not required profile setting.</u>
<u>OMX_VIDEO_VP8ProfileMain</u>	<u>VP8 Main profile</u>

- eLevel is the level used for VP8 encoding or decoding.

Table 4-69 shows the possible VP8 video level types in OMX\_VIDEO\_VP8LEVELTYPE.

Table 4-69: Supported VP8 Level Types

<u>Field Name</u>	<u>VP8 Level Descriptions</u>
<u>OMX_VIDEO_VP8LevelUnknown</u>	<u>Unknown, unused or not required setting</u>
<u>OMX_VIDEO_VP8Level_Version0</u>	<u>VP8 Level “Version 0”</u>
<u>OMX_VIDEO_VP8Level_Version1</u>	<u>VP8 Level “Version 1”</u>
<u>OMX_VIDEO_VP8Level_Version2</u>	<u>VP8 Level “Version 2”</u>
<u>OMX_VIDEO_VP8Level_Version3</u>	<u>VP8 Level “Version 3”</u>

In VP8 certain decoding tools are enabled or disabled based on the eLevel and higher level means less decoding complexity.

Table 4-70 shows which decoding tools are enabled or disabled.

**Table 4-70: Decoding Tools Usage Based on VP8 level**

Level	Reconstruction Filter	Loop Filter
“Version 0”	Bicubic	Normal
“Version 1”	Bilinear	Simple
“Version 2”	Bilinear	None
“Version 3”	None	None

- `nDCTPartitions` specifies the number of DCT coefficient data partitions within a compressed frame. Using more than 1 partition may allow more effective multi-threaded decoding.

Table 4-71 shows the possible values for `nDCTPartitions`.

**Table 4-71: nDCTPartitions Values**

Value	Description
0	1 DCT residual partition
1	2 DCT residual partitions
2	4 DCT residual partitions
3	8 DCT residual partitions

- `bErrorResilientMode` is a Boolean value used to indicate if error resilient mode is enabled. This mode prevents cumulative probability updates and is used in video telephony.

#### **4.3.19.2 Dependencies**

This parameter is only applicable when the port is configured for VP8.

#### **4.3.20 OMX\_VIDEO\_VP8REFERENCEFRAMETYPE**

`OMX_VIDEO_VP8REFERENCEFRAMETYPE` structure is used to configure the type of reference frames to be used while video encoding is in progress.

VP8 uses two encoding concepts:

##### **1) Frame coding type.**

There are only two types of frames in VP8, intraframes (key frames, I-frames) and interframes (prediction frames, P-frames). Frame coding type is controlled with `OMX_CONFIG_INTRAREFRESHVOPTYPE` structure.

##### **2) Reference frame buffers.**

VP8 uses three reference frame buffers called immediately previous frame, golden frame and alternate frame to predict blocks in an interframe. Every key frame is automatically golden frame and alternate frame. Optionally any interframe may replace the most recent golden frame and/or alternate frame.

OMX\_VIDEO\_VP8REFERENCEFRAMETYPE is defined as follows.

```
typedef struct OMX_VIDEO_VP8REFERENCEFRAMETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL nPreviousFrameRefresh;
    OMX_BOOL bGoldenFrameRefresh;
    OMX_BOOL bAlternateFrameRefresh;
    OMX_BOOL bUsePreviousFrame;
    OMX_BOOL bUseGoldenFrame;
    OMX_BOOL bUseAlternateFrame;
} OMX_VIDEO_VP8REFERENCEFRAMETYPE ;
```

#### 4.3.20.1 Parameters

The parameters for OMX\_VIDEO\_VP8REFERENCEFRAMETYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- bPreviousFrameRefresh is a Boolean value used to indicate if the next frame is used to refresh (update) the immediately previous frame.
- bGoldenFrameRefresh is a Boolean value used to indicate if the next frame is to be encoded as a golden reference frame.
- bAlternateFrameRefresh is a Boolean value used to indicate if the next frame is to be encoded as an alternate reference frame.
- bUsePreviousFrame is a Boolean value used to indicate if the immediately previous frame should be used for prediction.
- bUseGoldenFrame is a Boolean value used to indicate if the golden reference frame should be used for prediction.
- bUseAlternateFrame is a Boolean value used to indicate if the alternate reference frame should be used for prediction.

Table 4-72: Possible Ways to Refresh Reference Frames

<u>bPreviousFrameRefresh</u>	<u>bGoldenFrameRefresh</u>	<u>bAlternateFrameRefresh</u>	<u>Effect on Coding</u>
<u>OMX_FALSE</u>	<u>OMX_FALSE</u>	<u>OMX_FALSE</u>	Droppable frame. Usable for temporal scalability, as no future frames will use this frame as a reference. Frame is droppable if <u>bErrorResilientMode = OMX_TRUE</u> .

<u>OMX_FALSE</u>	<u>OMX_FALSE</u>	<u>OMX_TRUE</u>	<u>Alternate reference frame is updated by this frame.</u>
<u>OMX_FALSE</u>	<u>OMX_TRUE</u>	<u>OMX_FALSE</u>	<u>Golden reference frame is updated by this frame.</u>
<u>OMX_FALSE</u>	<u>OMX_TRUE</u>	<u>OMX_TRUE</u>	<u>Alternate reference frame and golden reference frame are updated by this frame.</u>
<u>OMX_TRUE</u>	<u>OMX_FALSE</u>	<u>OMX_FALSE</u>	<u>Immediately previous frame is updated by this frame.</u>
<u>OMX_TRUE</u>	<u>OMX_FALSE</u>	<u>OMX_TRUE</u>	<u>Immediately previous frame and alternate reference frame are updated by this frame.</u>
<u>OMX_TRUE</u>	<u>OMX_TRUE</u>	<u>OMX_FALSE</u>	<u>Immediately previous frame and golden reference frame are updated by this frame.</u>
<u>OMX_TRUE</u>	<u>OMX_TRUE</u>	<u>OMX_TRUE</u>	<u>Immediately previous frame, golden reference frame, and alternate reference frame are updated by this frame.</u>

#### 4.3.21 OMX\_VIDEO\_VP8REFERENCEFRAMEINFOTYPE

OMX\_VIDEO\_VP8REFERENCEFRAMEINFOTYPE structure is used to report the VP8 reference frame type while video decoding is in progress.

OMX\_VIDEO\_VP8REFERENCEFRAMEINFOTYPE is defined as follows.

```
typedef struct OMX_VIDEO_VP8REFERENCEFRAMEINFOTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bIsIntraFrame;
    OMX_BOOL bIsGoldenOrAlternateFrame;
} OMX_VIDEO_VP8REFERENCEFRAMEINFOTYPE;
```

##### 4.3.21.1 Parameters

The parameters for OMX\_VIDEO\_VP8REFERENCEFRAMEINFOTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- bIsIntraFrame is a Boolean value used to indicate if the frame is an Intra frame.
- bIsGoldenOrAlternateFrame is a Boolean value used to indicate if the frame is a golden frame or an alternate frame.

##### 4.3.21.2 Dependencies

The parameter may only be used to query the reference frame type at any time that the component is in the OMX\_StateExecuting state.

#### **4.3.194.3.22 OMX\_VIDEO\_CONFIG\_BITRATETYPE**

The video encoder's bit rate setting may be updated while the video encoder is actively encoding, the OMX\_VIDEO\_CONFIG\_BITRATETYPE structure contains the parameters for updating the video bit rate.

OMX\_VIDEO\_CONFIG\_BITRATETYPE is defined as follows.

```
typedef struct OMX_VIDEO_CONFIG_BITRATETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nEncodeBitrate;
} OMX_VIDEO_CONFIG_BITRATETYPE;
```

#### **4.3.19.14.3.22.1 Parameters**

The parameters for OMX\_VIDEO\_CONFIG\_BITRATETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nEncodeBitrate is the target bit rate for the video encoding in units of bits per second .

#### **4.3.204.3.23 OMX\_CONFIG\_FRAMERATETYPE**

The video encoder's frame rate setting may be updated while the video encoder is actively encoding, the OMX\_CONFIG\_FRAMERATETYPE structure contains the parameters for updating the video frame rate.

OMX\_CONFIG\_FRAMERATETYPE is defined as follows.

```
typedef struct OMX_CONFIG_FRAMERATETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 xEncodeFramerate;
} OMX_CONFIG_FRAMERATETYPE;
```

#### **4.3.20.14.3.23.1 Parameters**

The parameters for OMX\_CONFIG\_FRAMERATETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- xEncodeFramerate is the frame rate for the video encoding in units of frames per second. This value is represented in Q16 format

#### **4.3.214.3.24 OMX\_CONFIG\_INTRAREFRESHVOPTYPE**

The OMX\_CONFIG\_INTRAREFRESHVOPTYPE structure is used to force the next video frame to be encoded as an I-VOP.

OMX\_CONFIG\_INTRAREFRESHVOPTYPE is defined as follows.

```
typedef struct OMX_CONFIG_INTRAREFRESHVOPTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL IntraRefreshVOP;
} OMX_CONFIG_INTRAREFRESHVOPTYPE;
```

#### **4.3.214.3.24.1 Parameters**

The parameters for OMX\_CONFIG\_INTRAREFRESHVOPTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- IntraRefreshVOP is a Boolean value used to indicate if the next frame is to be encoded as an I VOP.

#### **4.3.224.3.25 OMX\_CONFIG\_MACROBLOCKERRORMAPTYPE**

The OMX\_CONFIG\_MACROBLOCKERRORMAPTYPE structure is used to force some of all of the macroblocks within the next video frame to be encoded as Intra macroblocks.

Typically the map of the macroblocks requested to be refreshed as intra macroblocks correlates to macroblock decoding errors encountered during a video telephony use case on the remote device.

OMX\_CONFIG\_MACROBLOCKERRORMAPTYPE is defined as follows.

```
typedef struct OMX_CONFIG_MACROBLOCKERRORMAPTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nErrMapSize;
    OMX_U8 ErrMap[1];
} OMX_CONFIG_MACROBLOCKERRORMAPTYPE;
```

#### **4.3.224.3.25.1 Parameters**

The parameters for OMX\_CONFIG\_MACROBLOCKERRORMAPTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.

- `nErrMapSize` is the size of the macroblock map containing the refresh information, this parameter is specified in units of bytes.
- `ErrMap` contains the map of the macroblocks within the frame that are to be refreshed as intra macroblocks. The array contains one or more bytes as indicated by the `nErrMapSize` field

The format of the macroblock map is a bit mapped string of values that corresponds to each macroblock within the video frame, when the bit value is set it indicates that the corresponding macroblock is to be refreshed as an intra macroblock.

As an example, a video frame having a resolution of 176x144 contains 99 macroblocks thus the macroblock map will contain 99 bit mapped values identifying each and every macroblock within the frame (the `nErrMapSize` parameter will contain a size of 13 – rounded up to the nearest byte boundary). Bit 0 of the macroblock map refers to macroblock 0 within the video frame, bit 1 refers to macroblock 1 and so on.

The error map information is cumulative between frames; it is to be cleared:

- Upon each `OMX_GetConfig` request.
- Each time an Intra Frame is detected. The error map information is to include any macroblock errors found within the Intra frame.

#### **4.3.22.24.3.25.2 Dependencies**

The parameter may only be used to get the macroblock error map information using `OMX_GetConfig` at any time that the component is in the `OMX_StateExecuting` state.

#### **4.3.22.34.3.25.3 Error Conditions**

On processing the `OMX_CONFIG_MACROBLOCKERRORMAPTYPE` structure, the following error conditions can occur:

- `OMX_ErrorMbErrorsInFrame` when macroblock errors are found within a frame.

When macroblock errors are encountered during the processing, the component will issue an `OMX_EventError` event with the value `OMX_ErrorMbErrorsInFrame` notifying the IL client of this occurrence.

#### **4.3.234.3.26 OMX\_PARAM\_MACROBLOCKSTYPE**

The `OMX_PARAM_MACROBLOCKSTYPE` structure is used to report the number of macroblocks available within the current video stream's frame.

`OMX_PARAM_MACROBLOCKSTYPE` is defined as follows.

```
typedef struct OMX_PARAM_MACROBLOCKSTYPE {
```

```
OMX_U32 nSize;
OMX_VERSIONTYPE nVersion;
OMX_U32 nPortIndex;
OMX_U32 nMacroblocks;
} OMX_PARAM_MACROBLOCKSTYPE;
```

#### **4.3.23.14.3.26.1 Parameters**

The parameters for OMX\_PARAM\_MACROBLOCKSTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- nMacroblocks is the number of macroblocks available within the video frame.

#### **4.3.23.24.3.26.2 Dependencies**

The parameter may only be used to query the number of macroblocks within the video frame using OMX\_GetParameter at any time that the component is in the OMX\_StateExecuting state.

#### **4.3.24.4.3.27 OMX\_CONFIG\_MBERRORREPORTINGTYPE**

The OMX\_CONFIG\_MBERRORREPORTINGTYPE structure is used to enable or disable the macroblock error reporting support.

The macroblock error map information is queried from the video decoder with OMX\_GetConfig using OMX\_IndexConfigVideoMacroBlockErrorMap and the OMX\_CONFIG\_MACROBLOCKERRORTYPE structure.

OMX\_CONFIG\_MBERRORREPORTINGTYPE is defined as follows.

```
typedef struct OMX_CONFIG_MBERRORREPORTINGTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnabled;
} OMX_CONFIG_MBERRORREPORTINGTYPE;
```

#### **4.3.24.14.3.27.1 Parameters**

The parameters for OMX\_CONFIG\_MBERRORREPORTINGTYPE are defined as follows.

- nPortIndex ~~is the read-only value containing the index of the portrepresents the port that this structure applies to.~~
- bEnabled is a Boolean value indicating to enable to disable the macroblock error reporting support.

#### **4.3.254.3.28 OMX\_VIDEO\_PARAM\_PROFILELEVELTYPE**

The OMX\_VIDEO\_PARAM\_PROFILELEVELTYPE structure is used to query the video encoders and decoders for their supported profiles and associated levels when used with the OMX\_IndexParamVideoProfileLevelSupported.

In addition the structure may also be used to query or set the profile and level of the video stream that is currently being processed, this is achieved using [OMX\\_IndexParamVideoProfileLevelCurrent](#).

[The codec information retrieved is dependent on the current coding format specified as per the port definition – OMX\\_VIDEO\\_PORTDEFINITIONTYPE or OMX\\_VIDEO\\_PARAM\\_PORTFORMATTYPE](#). The caller is required to type cast [eCodecType](#), [eProfile](#) and [eLevel](#) parameters to the proper data enumeration types prior to interpreting the parameter information. The type casting is to be based on the [eCompressionFormat](#) parameter defined by either [OMX\\_VIDEO\\_PORTDEFINITIONTYPE](#) or [OMX\\_VIDEO\\_PARAM\\_PORTFORMATTYPE](#).

[Some of the structure parameters may not be applicable or used for some of the coding types, refer to Table 4-73: Profile and Level Type Casting to understand the parameter usage versus coding type.](#)

OMX\_VIDEO\_PARAM\_PROFILELEVELTYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_PROFILELEVELTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 eProfile;  
    OMX_U32 eLevel;  
    OMX_U32 nProfileIndex;  
    OMX_U32 eCodecType;  
} OMX_VIDEO_PARAM_PROFILELEVELTYPE;
```

#### **4.3.25.14.3.28.1 Parameters**

The parameters for OMX\_VIDEO\_PARAM\_PROFILELEVELTYPE are defined as follows.

- [nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.](#)
- [eProfile is the profile setting as associated with the eCompressionFormat parameter.](#)  
[The caller is required to type cast both this parameter to the proper data enumeration types prior to interpreting the parameter information – refer to Table 4-73: Profile and Level Type Casting for the casting parameters.](#)

[Table 4-73: Profile and Level Type Casting](#)

<u>Coding Type</u>	<u>Codec Type</u>	<u>Profile Type</u>	<u>Level Type</u>
<u>OMX_VIDEO_CodingMPEG2</u>	<u>Not Applicable</u>	<u>OMX_VIDEO_MPEG2P FILETYPE</u>	<u>OMX_VIDEO_MPEG2L LEVELTYPE</u>
<u>OMX_VIDEO_CodingH263</u>	<u>Not Applicable</u>	<u>OMX_VIDEO_H263PR FILETYPE</u>	<u>OMX_VIDEO_H263LE LEVELTYPE</u>
<u>OMX_VIDEO_CodingMPEG4</u>	<u>Not Applicable</u>	<u>OMX_VIDEO_MPEG4P FILETYPE</u>	<u>OMX_VIDEO_MPEG4L LEVELTYPE</u>
<u>OMX_VIDEO_CodingWMV</u>	<u>OMX_VIDEO_WMVFORMATTYPE</u>	<u>OMX_VIDEO_WMVPRO FILETYPE</u>	<u>OMX_VIDEO_WMVLEV ELTYPE</u>
<u>OMX_VIDEO_CodingRV</u>	<u>Not Applicable</u>	<u>OMX_VIDEO_RVFORM ATTTYPE</u>	<u>Not Applicable</u>
<u>OMX_VIDEO_CodingAVC</u>	<u>Not Applicable</u>	<u>OMX_VIDEO_AVCPRO FILETYPE</u>	<u>OMX_VIDEO_AVCLEV ELTYPE</u>
<u>OMX_VIDEO_CodingVC1</u>	<u>Not Applicable</u>	<u>OMX_VIDEO_VC1PRO FILETYPE</u>	<u>OMX_VIDEO_VC1LEV ELTYPE</u>
<u>OMX_VIDEO_CodingVP8</u>	<u>Not Applicable</u>	<u>OMV_VIDEO_VP8PRO FILETYPE</u>	<u>OMX_VIDEO_VP8LEV ELTYPE</u>

- eLevel is the profile level setting as associated with the eCompressionFormat and eProfile parameters.

The caller is required to type cast both this parameter to the proper data enumeration types prior to interpreting the parameter information – refer to Table 4-73: Profile and Level Type Casting for the casting parameters.

The caller is required to type cast eCodecType both the eProfile and eLevel parameters to the proper data enumeration types prior to interpreting the parameter information. The type casting is to be based on the eCompressionFormat parameter defined as per the port definition. Shows the profile and level type casting parameters.

For OMX\_VIDEO\_CodingWMV coding type the profile and level definitions are:

Table 4-74: WMV Profile Types

<u>OMX_VIDEO_WMVPROFILETYPE Types</u>	<u>WMV Profile Descriptions</u>
<u>OMX_VIDEO_WMVProfileUnknown</u>	<u>Unknown, unused or not required setting.</u>
<u>OMX_VIDEO_WMVProfileSimple</u>	<u>WMV Simple Profile</u>
<u>OMX_VIDEO_WMVProfileMain</u>	<u>WMV Main Profile</u>
<u>OMX_VIDEO_WMVProfileAdvanced</u>	<u>WMV Advanced Profile</u>

Table 4-75: WMV Level Types

<u>OMX_VIDEO_WMVLEVELTYPE Types</u>	<u>WMV Level Descriptions</u>
<u>OMX_VIDEO_WMVLevelUnknown</u>	<u>Unknown, unused or not required setting.</u>
<u>OMX_VIDEO_WMVLevelLow</u>	<u>WMV Low level</u>
<u>OMX_VIDEO_WMVLevelMedium</u>	<u>WMV Medium Level</u>
<u>OMX_VIDEO_WMVLevelHigh</u>	<u>WMV High Level</u>
<u>OMX_VIDEO_WMVLevelL0</u>	<u>WMV L0 Level</u>
<u>OMX_VIDEO_WMVLevelL1</u>	<u>WMV L1 Level</u>
<u>OMX_VIDEO_WMVLevelL2</u>	<u>WMV L2 Level</u>
<u>OMX_VIDEO_WMVLevelL3</u>	<u>WMV L3 Level</u>
<u>OMX_VIDEO_WMVLevelL4</u>	<u>WMV L4 Level</u>

- nProfileIndex is used to enumerate the supported profiles. The caller specifies all fields and the OMX\_GetParameter call returns the value of the supported profile and level. The value of nProfileIndex goes from 0 to N-1, where N is the number of profiles supported by the port. The port does not need to report N as the caller can determine N by enumerating all the formats supported by the port. Each port shall support at least one profile. If there are no more profiles, OMX\_GetParameter returns OMX\_ErrorNoMore .

**Table 4-76: ProfileLevel Call Details**

Action	Index	Description
Query for supported profiles and levels	OMX_IndexParamVideoProfileLevel QuerySupported	Multiple calls with increasing values of <u>nProfileIndex</u> will enumerate the supported profiles until OMX_ErrorNoMore is returned. With each successful call, a supported profile will be identified with the maximum supported associated level setting.
Query the profile and level for the current stream	OMX_IndexParamVideoProfileLevel Current	eCompressionFormat, eProfile and eLevel will return the current stream's information. The <u>nProfileIndex</u> parameter is an ignored parameter.

Action	Index	Description
Configure the encoder to use a specific profile and level for the current stream	OMX_IndexParamVideoProfileLevelCurrent	eCompressionFormat, eProfile and eLevel will contain the requested settings to be used as part of the encoding. The nProfileIndex parameter is an ignored parameter.

- eCodecType is the format setting as associated with the eCompressionFormat parameter.

The caller is required to type cast both this parameter to the proper data enumeration types prior to interpreting the parameter information – refer to Table 4-73: Profile and Level Type CastingError! Reference source not found. for the casting parameters.

#### 4.3.25.24.3.28.2 Dependencies

The parameter using the index OMX\_IndexParamVideoProfileLevelCurrent may be queried using OMX\_GetParameter or set using OMX\_SetParameter at any time that the component is initialized.

The IL ClientIL client shall ignore any parameter identified in Table 4-60 as “Not Applicable” as any parameter specified in the table as “Not Applicable” has no associated information for the specified Coding Type.

For any table entry in Table 4-60 identified as “Not Applicable”, the IL Client shall ignore this parameter information since this information has no associated information for the coding type’s specification thus no suitable information can be identified.

#### 4.3.264.3.29 OMX\_VIDEO\_PARAM\_AVCSLICEFMO

The OMX\_VIDEO\_PARAM\_AVCSLICEFMO structure is used to enable and configure the Flexible Macroblock Ordering (FMO) slice modes within the AVC video encoder.

OMX\_VIDEO\_PARAM\_AVCSLICEFMO is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_AVCSLICEFMO {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U8 nNumSliceGroups;
    OMX_U8 nSliceGroupMapType;
    OMX_VIDEO_AVCSLICEMODETYPE eSliceMode;
```

```
} OMX_VIDEO_PARAM_AVCSLICEFMO;
```

#### 4.3.26.14.3.29.1 Parameters

The parameters for OMX\_VIDEO\_PARAM\_AVCSLICEFMO are defined as follows.

- nPortIndex is the read only value containing the index of the port represents the port that this structure applies to.
- nNumSliceGroups specifies the number of slice groups that can be supported in the encode session. This parameter is enabled when FMO mode is enabled, refer to OMX\_VIDEO\_PARAM\_AVCTYPE for enabling FMO mode support.

The setting information for this parameter is directly related to the functionality as specified within the ITU H.264/AVC specification and is dependent on the video profile currently in use.

The currently defined parameter range settings are listed in [Table 4-77](#).

**Table 4-77: AVC Parameter Range Settings**

Video Profile	Range
OMX_VIDEO_AVCPProfileBaseline	0 to 7
OMX_VIDEO_AVCPProfileMain	0
OMX_VIDEO_AVCPProfileExtended	0 to 7
OMX_VIDEO_AVCPProfileHigh	0
OMX_VIDEO_AVCPProfileHigh10	0
OMX_VIDEO_AVCPProfileHigh422	0
OMX_VIDEO_AVCPProfileHigh444	0

- nSliceGroupMapType specifies the type of slice groupings that is to be used during encoding.

The setting information for this parameter is directly related to the functionality as specified within the ITU H.264/AVC specification.

The currently defined parameter settings are:

**Table 4-78: Slice Group Map Type Values**

Slice Group Map Value	Description
0	Indicates interleaves slices.
1	Indicates a dispersed macroblock allocation
2	Indicates to explicitly assign a slice group to each macroblock in raster scan order
3	Indicates one or more “foreground” slice groups and a “leftover” slice group
4	Indicates changing slice groups.

Slice Group Map Value	Description
5	Indicates changing slice groups.
6	Indicates changing slice groups.

- eSliceMode specifies the type of slice that is to be used for encoding the frame.

**Table 4-79: Slice Mode Type Casting**

Slice Mode	AVC Slice Mode Description
OMX_VIDEO_SLICEMODE_AVCDefault	Normal frame encoding, one slice per frame
OMX_VIDEO_SLICEMODE_AVCMBSlice	NAL mode based on number of macroblocks per slice
OMX_VIDEO_SLICEMODE_AVCBytesSlice	NAL Mode based on number of bytes per slice.

#### **4.3.274.3.30 OMX\_VIDEO\_CONFIG\_AVCINTRAPERIOD**

The OMX\_VIDEO\_CONFIG\_AVCINTRAPERIOD structure is used to enable and configure the IDR and Intra periodicity for the AVC encoder during an encoding session.

OMX\_VIDEO\_CONFIG\_AVCINTRAPERIOD is defined as follows.

```
typedef struct OMX_VIDEO_CONFIG_AVCINTRAPERIOD {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nIDRPeriod;
    OMX_U32 nPFrames;
} OMX_VIDEO_CONFIG_AVCINTRAPERIOD;
```

#### **4.3.27.14.3.30.1 Parameters**

The parameters for OMX\_VIDEO\_CONFIG\_AVCINTRAPERIOD are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nIDRPeriod defines the periodicity of IDR occurrence. This specifies coding a frame as IDR after a specific number every nPFrames of intra frames. The periodicity of the intra frame coding is specified by nPFrames. If nIDRPeriodthis parameter is set to 0, only the first frame of the encode session is an IDR frame.
- nPFrames specifies coding of a frame as Intra (non-inclusive of the first frame) after every nPFrames of Inter frames.

#### **4.3.284.3.31 OMX\_VIDEO\_CONFIG\_NALSIZE**

The OMX\_VIDEO\_CONFIG\_NALSIZE structure is used to specify the size of a NAL unit for the AVC encoder during an encoding session.

`OMX_VIDEO_CONFIG_NALSIZE` is defined as follows.

```
typedef struct OMX_VIDEO_CONFIG_NALSIZE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nNaluBytes;
} OMX_VIDEO_CONFIG_NALSIZE;
```

#### **4.3.28.14.3.31.1 Parameters**

The parameters for `OMX_VIDEO_CONFIG_NALSIZE` are defined as follows.

- `nPortIndex` is the read-only value containing the index of the port represents the port that this structure applies to.
- `nNaluBytes` specifies the number of bytes of data to be contained in the current NAL Units.

### **4.3.32 OMX\_NALSTREAMFORMATTYPE**

The `OMX_NALSTREAMFORMATTYPE` structure is used to specify the NAL unit format and its associated size.

`OMX_NALSTREAMFORMATTYPE` is defined as follows.

```
typedef struct OMX_NALSTREAMFORMATTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_NALUFORMATSTYPE eNaluFormat;
} OMX_NALSTREAMFORMATTYPE;
```

#### **4.3.32.1 Parameters**

The parameters for `OMX_NALSTREAMFORMATTYPE` are defined as follows.

- `nPortIndex` represents the port that this structure applies to.
- `eNaluFormat` indicates the format of the NALU. Refer to Table 4-80 for a listing of the various formats. This parameter contains bit-mapped values as defined by Table 4-80.

The default mode of operation shall be `OMX_NaluFormatStartCodes`.

<u>NALU Format</u>	<u>Description</u>
<code>OMX_NaluFormatStartCodes</code>	NALUs separated by Start Codes (ITU-T H.264/ISO 14496-10 Annex B)
<code>OMX_NaluFormatOneNaluPerBuffer</code>	One NALU per buffer. Multiple NALUs in the same buffer are forbidden
<code>OMX_NaluFormatOneByteInterleaveLength</code>	NALU separated by 1-byte interleaved length fields

<u>OMX_NaluFormatTwoByteInterleaveLength</u>	NALU separated by 2-byte interleaved length fields
<u>OMX_NaluFormatFourByteInterleaveLength</u>	NALU separated by 4-byte interleaved length fields

**Table 4-80: NALU Formats**

Payload Packaging Options for cases when Start Codes are not in use:

A buffer containing a single NAL unit appears as:

<NAL Size X bytes><NAL unit>

A buffer containing multiple NALs unit appears as:

<NAL Size X bytes><NAL unit><NAL Size X bytes><NAL unit>

<NAL Size X Bytes> is the number of bytes indicating the size of the NAL unit payload

In case of “OMX\_NaluFormatOneNaluPerBuffer” NALU format, only one full or partial <NAL unit> payload data without start codes or NAL sizes is present in the buffer.

#### **4.3.32.2 Functionality**

In order for an OpenMAX IL client to properly configure a component graph to consume the stream, it needs to be able to query the components to determine:

- The stream packaging format (Source Component – component emitting the NALU payload)
- The stream formats supported by the component consuming the NALU payload.

The determination of the NALU formatting shall be queried via the source components that will be emitting the stream content, for example Demuxer Components. These components will only have access to this formatting information when it has been given the opportunity to parse the source content, typically achieved when in OMX\_StateExecuting state. Utilizing the auto-detection support, the IL client will be able to query this information after the component issues the OMX\_EventPortFormatDetected event.

The IL client shall use

OMX\_GetParameter(OMX\_IndexParamNalStreamFormat) on the source component's output port to query the native NALU packaging format within the embedded stream.

The IL client shall use

OMX\_GetParameter(OMX\_IndexParamNalStreamFormatSupported) on the source component's output port to query the NALU packaging formats supported – the nNaluFormat parameter shall return all the formats supported or'ed together.

The IL client shall use

OMX\_GetParameter(OMX\_IndexParamNalStreamFormatSupported) on the consumer component's input port to query the NALU packaging format supported – the nNaluFormat parameter shall return all the formats supported or'ed together.

In the case where a consumer component's input port does not support the NAL stream format selection, the responsibility of formatting the stream payload reverts to the source component. A source component shall support the ability to emit the NALU payload in either configurable option, however it is not mandated that the component shall support the ability to package multiple NALUs within a single buffer – although this is highly recommended.

Note: Configuring a source component to format the NALU payload in a format that is non-native to the stream's embedded format may incur a performance penalty.

The IL client shall use

OMX\_SetParameter(OMX\_IndexParamNalStreamFormatSelect) on a source component's output port to configure it to the appropriate setting.

In the case where a consumer component's input port is capable of supporting the native NALU packaging format within the embedded stream but differs from the default OMX\_NaluFormatStartCodes mode, the IL client may configure the consumer component's input port instead of the source component's output port to consume the stream. The IL client shall use

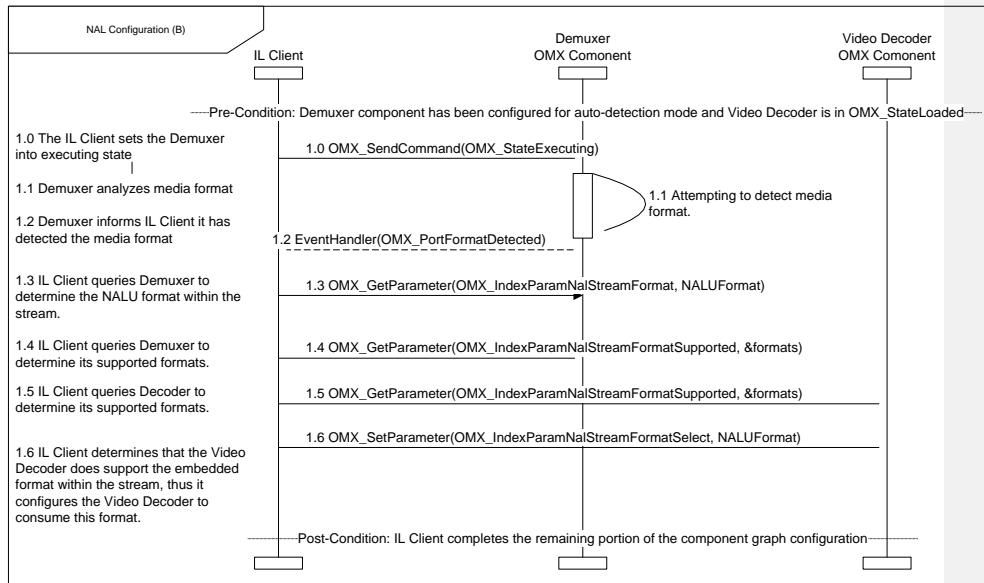
OMX\_SetParameter(OMX\_IndexParamNalStreamFormatSelect) on a consumer component's input port to configure it to the appropriate setting.

#### **4.3.28.3 Call Sequence Examples**

This section provides various examples that may be encountered.

Figure 4-10 shows the case when a Video Decoder supports the NALU configuration support within the embedded stream.

[Figure 4-11 shows the case when a Video Decoder does not support the NALU configuration within the stream, the IL client configures the Demuxer to emit a format supported by the Video Decoder \(e.g. NALU using Start Code - ITU-T H.264\ISO 14496-10 Annex B Specification\).](#)



[Figure 4-10: NALU Formatting Supported By Video Decoder](#)

The sequence starts with a Pre-Condition that the IL client has configured the output port formats (e.g. OMX\_IndexParamVideoPortFormat) of the Demuxer to auto detect.

The IL client commands the Demuxer component to transition into executing state – Step 1.0.

The Demuxer reads and parses the media content until it is able to detect the media container format – Step 1.1.

The Demuxer component detects the media format and notifies the IL client via an event callback – Step 1.2.

At this point, the Demuxer component is capable of determining the native NALU stream formatting within the embedded container. The IL client queries this information from the Demuxer – Step 1.3.

IL client queries the Demuxer to determine its supported formats (not a required step, shown for completeness) – Step 1.4.

IL client queries the Video Decoder to determine its supported formats – Step 1.5.

The IL client determines that the Video Decoder is capable of supporting the format within the stream. The IL client configures the Video Decoder to consume this format – Step 1.6.

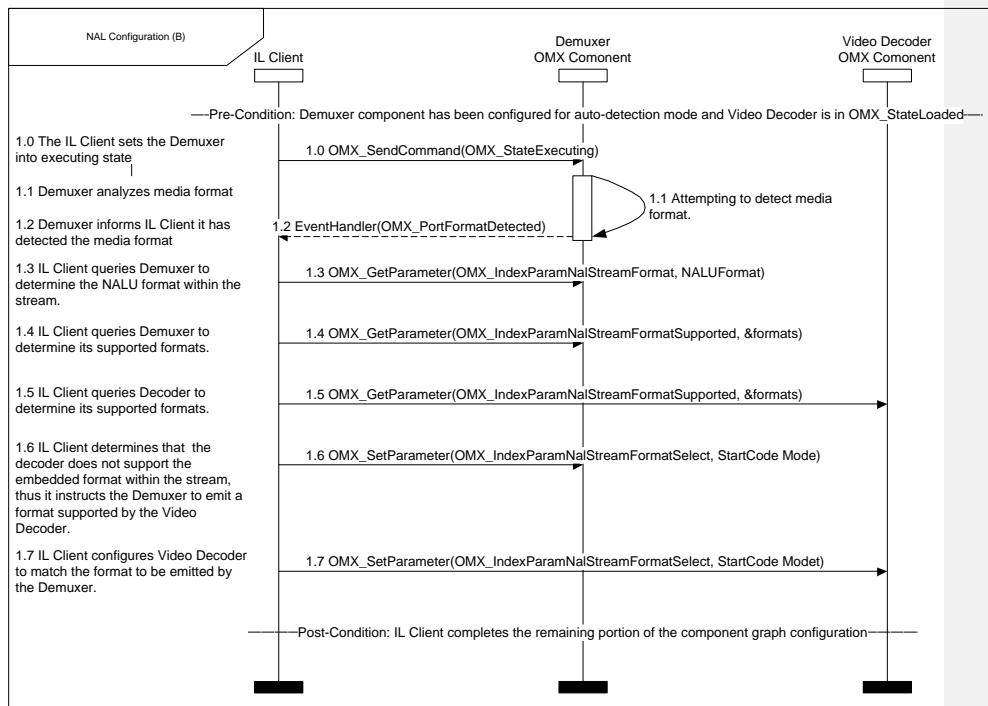


Figure 4-11: NALU Formatting Not Supported By Video Decoder

The sequence starts with a Pre-Condition that the IL client has configured the output port formats (e.g. OMX\_IndexParamVideoPortFormat) of the Demuxer to auto detect.

The IL client commands the Demuxer component to transition into executing state – Step 1.0.

The Demuxer reads and parses the media content until it is able to detect the media container format – Step 1.1.

The Demuxer component detects the media format and notifies the IL client via an event callback – Step 1.2.

At this point, the Demuxer component is capable of determining the native NALU stream formatting within the embedded container. The IL client queries this information from the Demuxer – Step 1.3.

IL client queries the Demuxer to determine its supported formats (not a required step, shown for completeness) – Step 1.4.

IL client queries the Video Decoder to determine its supported formats – Step 1.5.

The IL client determines that the Video Decoder does not support the format within the stream. The IL client configures the Demuxer to emit a format supported by the Video Decoder – Step 1.6.

The IL client configures the Video Decoder to consume the format to be emitted by the Demuxer – Step 1.7.

#### **4.3.33 OMX VIDEO PARAM VC1TYPE**

OMX\_VIDEO\_PARAM\_VC1TYPE contains common standard video codec parameters that are used with VC-1 format including VC-1 Simple profile, VC-1 Main Profile and VC-1 Advanced profile.

OMX\_VIDEO\_PARAM\_VC1TYPE is defined as follows.

```
typedef struct OMX_VIDEO_PARAM_VC1TYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_VIDEO_VC1PROFILETYPE eProfile;  
    OMX_VIDEO_VC1LEVELTYPE eLevel;  
} OMX_VIDEO_PARAM_VC1TYPE;
```

##### **4.3.33.1 Parameters**

The parameters for OMX\_VIDEO\_PARAM\_VC1TYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- eProfile is the enumerated value representing the profile of the VC-1 data stream. Table 4-81 shows the possible VC-1 video profile types for OMX\_VIDEO\_VC1PROFILETYPE.

**Table 4-81 : Supported VC-1 Profile Types**

<u>Field Name</u>	<u>VC-1 Profile Descriptions</u>
<u>OMX_VIDEO_VC1ProfileUnknown</u>	<u>Unknown, unused or not required setting.</u>
<u>OMX_VIDEO_VC1ProfileUnused</u>	<u>Profile unused or unknown</u>
<u>OMX_VIDEO_VC1ProfileSimple</u>	<u>Simple Profile</u>
<u>OMX_VIDEO_VC1ProfileMain</u>	<u>Main Profile</u>

<u>Field Name</u>	<u>VC-1 Profile Descriptions</u>
<u>OMX_VIDEO_VC1ProfileAdvanced</u>	Advanced Profile

- eLevel is the processing level of a particular VC-1 profile. Table 4-82 shows the possible VC-1 video level types in OMX\_VIDEO\_VC1LEVELTYPE.

Table 4-82 : Supported VC-1 Level Types

<u>Field Name</u>	<u>VC-1 Level Descriptions</u>
<u>OMX_VIDEO_VC1LevelUnknown</u>	<u>Unknown, unused or not required setting.</u>
<u>OMX_VIDEO_VC1LevelUnused</u>	<u>Level unused or unknown</u>
<u>OMX_VIDEO_VC1LevelLow</u>	<u>Simple or Main Profile - Low Level</u>
<u>OMX_VIDEO_VC1LevelMedium</u>	<u>Simple or Main Profile - Medium Level</u>
<u>OMX_VIDEO_VC1LevelHigh</u>	<u>Main Profile - High Level</u>
<u>OMX_VIDEO_VC1Level0</u>	<u>Level 0 - Advanced Profile</u>
<u>OMX_VIDEO_VC1Level1</u>	<u>Level 1 - Advanced Profile</u>
<u>OMX_VIDEO_VC1Level2</u>	<u>Level 2 - Advanced Profile</u>
<u>OMX_VIDEO_VC1Level3</u>	<u>Level 3 - Advanced Profile</u>
<u>OMX_VIDEO_VC1Level4</u>	<u>Level 4 - Advanced Profile</u>

#### 4.3.33.2 VC-1 Codec Configuration Data

##### 4.3.33.2.1 VC-1 Codec Configuration Data and Frame Layer Data for Main and Simple Profile

In case of VC-1 Simple and Main profile, the codec configuration data passed to the VC-1 decoder component shall be in the format specified by Table 265 of Annex L.2. of VC-1 specification, where STRUCT\_A, STRUCT\_B and STRUCT\_C of Table 265 are defined by Tables 260, 261, 263 of Annex J of VC-1 specification respectively. In case of VC-1 Simple and Main profile, in addition to codec configuration data, valid data in OMX buffer payloads (i.e. data pointed to by “pBuffer+nOffset” parameters of OMX buffer headers) provided to VC-1 decoder component shall contain compressed frame data corresponding to only FRAMEADATA portion of frame layer data of Table 266 of Annex L.3. of VC-1 specification.

##### 4.3.33.3 VC-1 Advanced Profile Start Codes

In case of VC-1 Advanced Profile, start codes are a part of the VC-1 bitstream, as defined in Annex G of VC-1 specification. As a default, the source component or IL client shall

provide VC-1 start codes to the VC-1 decoder component and shall not attempt to remove them from the bitstream, and the VC-1 decoder component shall support VC-1 Advanced Profile start codes. The source component or IL client may negotiate with the VC-1 decoder component a different format of VC-1 Advanced Profile bitstream and use such a format if both sides agree to it. However, as a minimum, both the VC-1 decoder component and the source component or IL client shall support the default VC-1 Advanced Profile bitstream format with embedded start codes.

#### **4.3.33.4 Handling of Multiple VC-1 Streams**

In the case where ASF content contains multiple VC-1 streams, each stream may require its own codec configuration. Also, in the case of multiple VC-1 streams, the source component or IL client may send data buffers that contain data from one stream, then switch to sending data buffers that contain data from a different stream to the VC-1 decoder component.

In case of such stream-switching, the source component or IL client shall ensure that the VC-1 decoder component receives the codec configuration data for the appropriate stream in-band, such that codec configuration data for the new stream immediately precedes the bitstream data for that stream (e.g. the source component or IL client ensure this by sending a buffer containing the codec configuration data for the new stream prior to sending any data buffers that contain the new bitstream data). This enables the VC-1 decoder component to reconfigure the decoder and correctly start processing data that belongs to the appropriate stream. Note that a VC-1 decoder component is not expected to handle multiple streams in parallel, since data that belongs to different streams is provided sequentially to the VC-1 decoder component (i.e. stream 1 codec configuration data, followed by stream 1 frame data, followed by stream 2 codec configuration data, followed by stream 2 frame data, etc.).

#### **4.3.33.5 Dependencies**

This parameter is only applicable when the port is configured for VC-1 video.

### **4.3.34 OMX VIDEO INTRAPERIODTYPE**

Intra-frame period is a parameter that may need to be updated during a video encoding session.

OMX\_VIDEO\_INTRAPERIODTYPE is a structure used to control this behavior.

OMX\_VIDEO\_INTRAPERIODTYPE is defined as follows.

```
typedef struct OMX_VIDEO_INTRAPERIODTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nIDRPeriod;
    OMX_S32 nPFrames;
    OMX_S32 nBFrames;
```

```
} OMX_VIDEO_INTRAPERIODTYPE;
```

#### 4.3.34.1 Parameters

The parameters for OMX\_VIDEO\_INTRAPERIODTYPE are defined as follows:

- nPortIndex represents the port that this structure applies to.
- nIDRPeriod is a value that defines the periodicity of IDR occurrence. This specifies coding a frame as IDR after a specific number of intra frames. The periodicity of intra-frame coding is specified by the nPFrames and nBFrames. If nIDRPeriod is set to 0, only the first frame of the encode session is an IDR frame. This field is used only for codecs that support IDR period, such as AVC.
- nPFrames is a value that specifies the number of P frames between each I Frame. The value less than 0 in this field shall be considered as “do-not-care” value.
- nBFrames is a value that specifies the number of B frames between each I Frame. Not all codec-profile types support configuring the presence of B Frames. This setting would be ignored for such codecs/profiles. The value less than 0 in this field shall be considered as “do-not-care” value.

## 4.4 Image

This section describes the parameter and configuration details for components and ports in the image domain. These parameter and configuration details are specified in the OMX\_Image.h header file.

### 4.4.1 Parameter and Configuration Indices

The header OMX\_Index.h contains the enumeration OMX\_INDEXTYPE, which contains all standard index values used core functions OMX\_GetParameter, OMX\_SetParameter, OMX\_GetConfig, and OMX\_SetConfig. Table 4-78 shows the index values that relate to imaging.

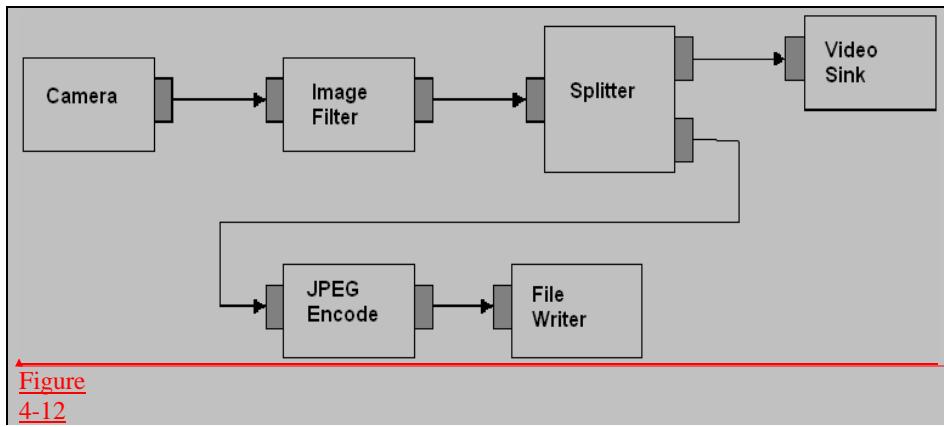
Table 4-78: Image Indices

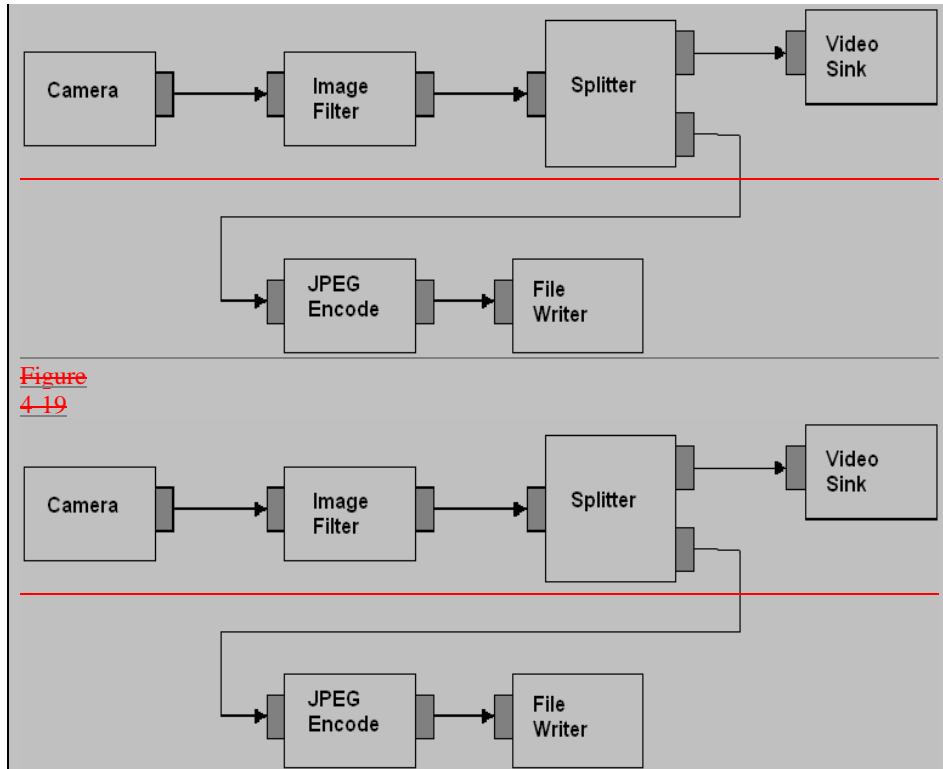
OpenMAX IL Indices (OMX_Index.h)	Corresponding OpenMAX IL Image Structures (OMX_Image.h)
OMX_IndexParamImagePortFormat	OMX_IMAGE_PARAM_PORTFORMATTYPE
OMX_IndexParamImageInit	OMX_PORT_PARAM_TYPE
OMX_IndexParamFlashControl	OMX_IMAGE_PARAM_FLASHCONTROLTYP
OMX_IndexConfigFlashControl	OMX_IMAGE_PARAM_FLASHCONTROLTYP
OMX_IndexConfigFocusControl	OMX_IMAGE_CONFIG_FOCUSCONTROLTYP
OMX_IndexParamQFactor	OMX_IMAGE_PARAM_QFACTORTYP

<b>OpenMAX IL Indices (OMX_Index.h)</b>	<b>Corresponding OpenMAX IL Image Structures (OMX_Image.h)</b>
<code>OMX_IndexParamQuantizationTable</code>	<code>OMX_IMAGE_PARAM_QUANTIZATIONTABLETYPE</code>
<code>OMX_IndexParamHuffmanTable</code>	<code>OMX_IMAGE_PARAM_HUFFMANTABLETYPE</code>
<code>OMX_IndexConfigFlickerRejection</code>	<code>OMX_CONFIG_FLICKERREJECTIONTYPE</code>
<code>OMX_IndexConfigImageHistogram</code>	<code>OMX_IMAGE_HISTOGRAMTYPE</code>
<code>OMX_IndexConfigImageHistogramData</code>	<code>OMX_IMAGE_HISTOGRAMDATATYPE</code>
<code>OMX_IndexConfigImageHistogramInfo</code>	<code>OMX_IMAGE_HISTOGRAMINFOTYPE</code>
<code>OMX_IndexConfigFileFormat</code>	<code>OMX_CONFIG_FILEFORMATTYPE</code>
<code>OMX_IndexConfigImageCaptureStarted</code>	<code>OMX_PARAM_U32TYPE</code>
	<i>This index allows receiving precise information about the moment when image to be captured is started to be exposed. The unsigned 32 bits value is used to count the number this situation occurs. This counter is initialized with 0.</i>
<code>OMX_IndexConfigImageCaptureEnded</code>	<code>OMX_PARAM_U32TYPE</code>
	<i>This index allows receiving precise information about the moment when image to be captured is finalized being exposed. The unsigned 32 bits value is used to count the number this situation occurs. This counter is initialized with 0.</i>

For example, `OMX_IndexParamImagePortFormat` index is used with `OMX_GetParameter` and `OMX_SetParameter` to access `OMX_IMAGE_PARAM_PORTFORMATTYPE`.

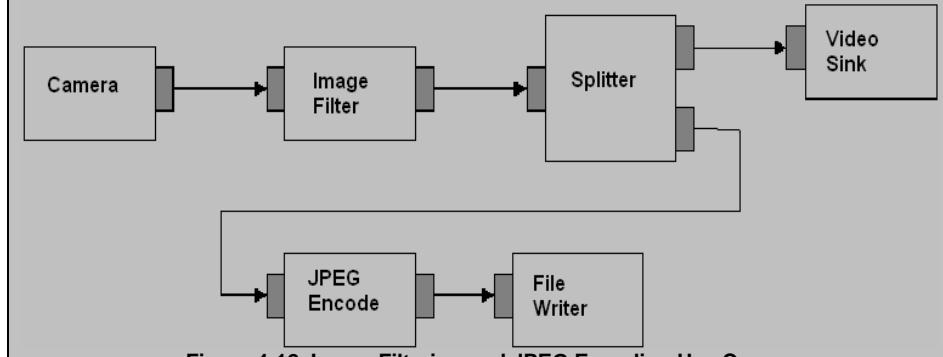
#### 4.4.24.4.1 Image Use Case Example





**Figure  
4-19**

**Figure 4-6** depicts one possible set of tunneled components and associated ports to implement a JPEG encoder with pre- and post-processing. This use case encodes an image to a file while allowing a preview of the captured image via a display.



**Figure 4-12. Image Filtering and JPEG Encoding Use Case**

Formatted: Font: Not Bold

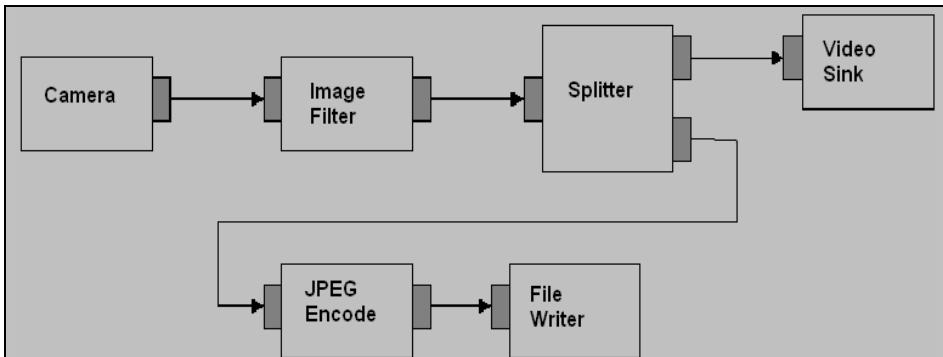


Figure  
4-12

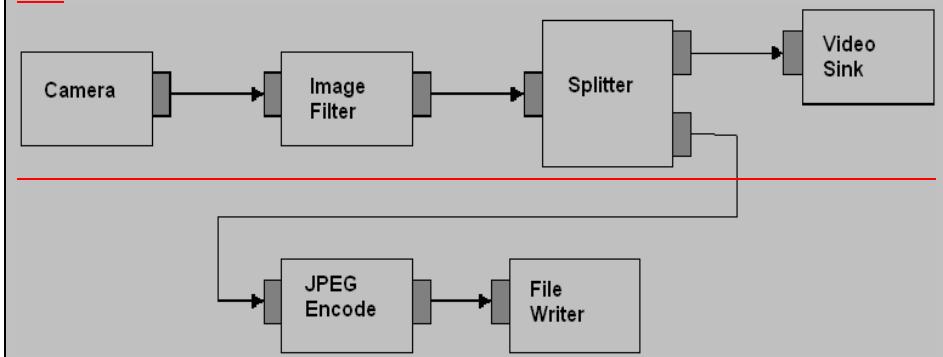


Figure  
4-19

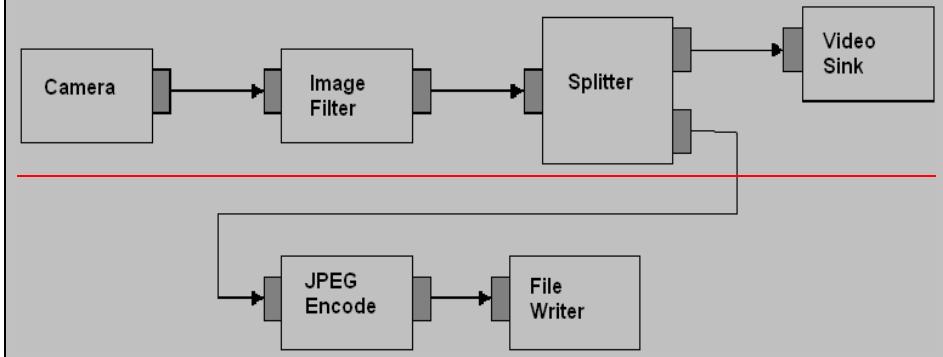


Figure 4-6 shows six components, namely the camera, the image filter, the splitter, the JPEG encoder, the file writer, and the image sink.

#### **4.4.2 Parameter and Configuration Indices**

The header OMX\_Index.h contains the enumeration OMX\_INDEXTYPE, which contains all standard index values used core functions OMX\_GetParameter, OMX\_SetParameter, OMX\_GetConfig, and OMX\_SetConfig, shows the index values that relate to imaging.

Field Code Changed

**Table 4-83: Image Indices**

<b>OpenMAX IL Indices (OMX_Index.h)</b>	<b>Corresponding OpenMAX IL Image Structures (OMX_Image.h)</b>
<u>OMX_IndexParamPortDefinition</u>	<u>OMX_PARAM_PORTDEFINITIONTYPE</u> with <u>OMX_IMAGE_PORTDEFINITIONTYPE</u>
<u>OMX_IndexParamImagePortFormat</u>	<u>OMX_IMAGE_PARAM_PORTFORMATTYPE</u>
<u>OMX_IndexParamImageInit</u>	<u>OMX_PORT_PARAM_TYPE</u>
<u>OMX_IndexParamFlashControl</u>	<u>OMX_IMAGE_PARAM_FLASHCONTROLTYP</u>
<u>OMX_IndexConfigFlashControl</u>	<u>OMX_IMAGE_PARAM_FLASHCONTROLTYP</u>
<u>OMX_IndexConfigFocusControl</u>	<u>OMX_IMAGE_CONFIG_FOCUSCONTROLTYP</u>
<u>OMX_IndexParamQFactor</u>	<u>OMX_IMAGE_PARAM_QFACTORTYP</u>
<u>OMX_IndexParamQuantizationTable</u>	<u>OMX_IMAGE_PARAM_QUANTIZATIONTABLETYP</u>
<u>OMX_IndexParamHuffmanTable</u>	<u>OMX_IMAGE_PARAM_HUFFMANTABLETYP</u>
<u>OMX_IndexConfigFlickerRejection</u>	<u>OMX_CONFIG_FLICKERREJECTIONTYP</u>
<u>OMX_IndexConfigImageHistogram</u>	<u>OMX_IMAGE_HISTOGRAMTYP</u>
<u>OMX_IndexConfigImageHistogramData</u>	<u>OMX_IMAGE_HISTOGRAMDATATYP</u>
<u>OMX_IndexConfigImageHistogramInfo</u>	<u>OMX_IMAGE_HISTOGRAMINFOTYP</u>
<u>OMX_IndexConfigFileFormat</u>	<u>OMX_CONFIG_FILEFORMATTYP</u>
<u>OMX_IndexConfigImageCaptureStarted</u>	<u>OMX_PARAM_U32TYP</u>
<u>OMX_IndexConfigImageCaptureEnded</u>	<u>OMX_PARAM_U32TYP</u>

For example, OMX\_IndexParamImagePortFormat index is used with OMX\_GetParameter and OMX\_SetParameter to access OMX\_IMAGE\_PARAM\_PORTFORMATTYPE.

#### **4.4.3 OMX\_IMAGE\_PORTDEFINITIONTYPE**

OMX\_IMAGE\_PORTDEFINITIONTYPE is the data structure that is used to define an image path. The number of image paths for input and output will vary by the type of the image component:

- Input (also known as source) has zero inputs and one output.
- Splitter has one input and two or more outputs.
- Processing element has one input and one output.
- Mixer has two or more inputs and one output.
- Output (also known as sink) has one input and zero outputs.

The OMX\_IMAGE\_PORTDEFINITIONTYPE structure can query the current or default definition of an image port or set the definition of an image port for a component. The OMX\_IMAGE\_PORTDEFINITIONTYPE structure is included as part of the OMX\_PARAM\_PORTDEFINITIONTYPE structure, it is accessed via the OMX\_GetParameter function or the OMX\_SetParameter function using the OMX\_IndexParamPortDefinition index.

OMX\_IMAGE\_PORTDEFINITIONTYPE is defined as follows.

```
typedef struct OMX_IMAGE_PORTDEFINITIONTYPE {
    OMX_STRING eMIMEType;
    OMX_NATIVE_DEVICETYPE pNativeRender;
    OMX_U32 nFrameWidth;
    OMX_U32 nFrameHeight;
    OMX_S32 nStride;
    OMX_U32 nSliceHeight;
    OMX_BOOL bFlagErrorConcealment;
    OMX_IMAGE_CODINGTYPE eCompressionFormat;
    OMX_COLOR_FORMATTYPE eColorFormat;
    OMX_NATIVE_WINDOWTYPE pNativeWindow;
} OMX_IMAGE_PORTDEFINITIONTYPE;
```

#### 4.4.3.1 Parameters

The parameters for OMX\_IMAGE\_PORTDEFINITIONTYPE are defined as follows.

- **eMIMEType** is the multipurpose Internet mail extensions (MIME) type of data on the port. If a MIME type string buffer is not supplied this parameter shall be set to NULL.
- **pNativeRender** is the read-only platform specific reference for a display synchronization; otherwise this field is 0. This parameter is ignored on OMX\_SetParameter calls.
- **nFrameWidth** is the width of frame to be used on the port if uncompressed format is used. Use 0 for unknown, no preference, or variable.
- **nFrameHeight** is the height of the frame to be used on the port if uncompressed format is used. Use 0 for unknown, no preference, or variable.
- **nStride** is a field containing the number of bytes per span of an image, which indicates the number of bytes to get from span N to span N+1. A negative value for nStride indicates the data is stored bottom-to-top instead of top-to-bottom.

Normally the stride parameter is determined by the component, there are cases however when the stride parameter may need to be updated based on external buffer stride requirements.

An example of such a case includes when IL clients submit buffers to the component for processing, the IL client may have differing stride requirements from the component port.

By allowing the flexibility for the stride to be modified, the component and IL client may negotiate a common stride setting to suit each other needs and in turn possibly improve the performance of processing the buffer.

- `nSliceHeight` is a read-only field containing the slice height parameter used when processing uncompressed image data. Buffers received on the port shall contain integer multiples of slices. For more information on minimum buffer payload for uncompressed data, see section 4.2.2.
- `bFlagErrorConcealment` is a flag indicating that the OpenMAX IL component supports error concealment. This flag is returned by a component upon invoking `OMX_SetParameter`; it is ignored on `OMX_SetParameter` calls.
- `eCompressionFormat` is the enumeration describing the compression format used on the port. When `OMX_IMAGE_CodingUnused` is specified, the `eColorFormat` field is valid. [Table 4-84](#) shows the supported image compression formats.

**Table 4-84: Supported Image Compression Formats**

Field Name	Compression Format Description	Reference to Standard
<code>OMX_IMAGE_CodingUnused</code>	No coding applied, use <code>eColorFormat</code>	Not available
<code>OMX_IMAGE_CodingAutoDetect</code>	Auto detection by the OpenMAX IL component	Not available
<code>OMX_IMAGE_CodingJPEG</code>	JPEG/JFIF image format	<a href="#">JPEG</a>
<code>OMX_IMAGE_CodingJPEG2K</code>	JPEG 2000 image format	<a href="#">JPEG2K</a>
<code>OMX_IMAGE_CodingEXIF</code>	EXIF image format	<a href="#">EXIF</a>
<code>OMX_IMAGE_CodingTIFF</code>	TIFF image format	<a href="#">TIFF</a>
<code>OMX_IMAGE_CodingGIF</code>	Graphics image format	<a href="#">GIF</a>
<code>OMX_IMAGE_CodingPNG</code>	PNG image format	<a href="#">PNG</a>
<code>OMX_IMAGE_CodingLZW</code>	LZW image format	<a href="#">LZW</a>
<code>OMX_IMAGE_CodingBMP</code>	Windows Bitmap format	<a href="#">BMP</a>
<code>OMX_IMAGE_CodingWEBP</code>	<a href="#">WebP image format</a>	<a href="#">WebP</a>
<code>OMX_IMAGE_CodingMax</code>	<a href="#">Maximum value</a>	Not available

- `eColorFormat` is the decompressed color format used for the port. This field is valid only when the `eCompressionFormat` field is set to `OMX_IMAGE_CodingUnused`.
- `pNativeWindow` is a platform specific reference for a windows object when being processed within as part of a video sink component, otherwise this field is 0 and ignored.

#### **4.4.4 OMX\_IMAGE\_PARAM\_PORTFORMATTYPE**

OMX\_IMAGE\_PARAM\_PORTFORMATTYPE is used to enumerate the various data input/output format supported by the port.

OMX\_IMAGE\_PARAM\_PORTFORMATTYPE is defined as follows.

```
typedef struct OMX_IMAGE_PARAM_PORTFORMATTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nIndex;
    OMX_IMAGE_CODINGTYPE eCompressionFormat;
    OMX_COLOR_FORMATTYPE eColorFormat;
} OMX_IMAGE_PARAM_PORTFORMATTYPE;
```

##### **4.4.4.1 Parameters**

The parameters for OMX\_IMAGE\_PARAM\_PORTFORMATTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nIndex indicates the enumeration index for the format from 0x0 to N-1.
- eCompressionFormat is an enumeration describing the compression format used on the port. When OMX\_IMAGE\_CodingUnused is specified, the eColorFormat field is valid. For enumerations regarding OMX\_IMAGE\_CODINGTYPE, see [Table 4-84](#).
- eColorFormat is the decompressed color format used for the port. This field is valid only when the eCompressionFormat field is set to OMX\_IMAGE\_CodingUnused. For enumerations on OMX\_COLOR\_FORMATTYPE, see section 4.2.

#### **4.4.5 OMX\_IMAGE\_PARAM\_FLASHCONTROLTYP**

The OMX\_IMAGE\_PARAM\_FLASHCONTROLTYP structure defines the mode of operation for flash control and configuration.

OMX\_IMAGE\_PARAM\_FLASHCONTROLTYP is defined as follows.

```
typedef struct OMX_IMAGE_PARAM_FLASHCONTROLTYP {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_IMAGE_FLASHCONTROLTYP eFlashControl;
} OMX_IMAGE_PARAM_FLASHCONTROLTYP;
```

##### **4.4.5.1 Parameters**

The parameters for OMX\_IMAGE\_PARAM\_FLASHCONTROLTYP are defined as follows.

- nPortIndex is the read only value containing the index of the portrepresents the port that this structure applies to.
- eFlashControl is an enumeration for the flash control modes. [Table 4-85](#) shows the supported image flash controls.

**Table 4-85: Supported Image Flash Controls**

Field Name	Flash Control Description
OMX_IMAGE_FlashControlOn	Strobe at every shot
OMX_IMAGE_FlashControlOff	Strobe off
OMX_IMAGE_FlashControlAuto	Strobe according to environment light
OMX_IMAGE_FlashControlRedEyeReduction	Pre-shot strobes
OMX_IMAGE_FlashControlFillin	Flash for background/foreground effect
OMX_IMAGE_FlashControlTorch	Flash is always on
<b>OMX_IMAGE_FlashControlMax</b>	<b>Maximum value</b>

#### 4.4.6 OMX\_IMAGE\_CONFIG\_FOCUSCONTROLTYPE

OMX\_IMAGE\_CONFIG\_FOCUSCONTROLTYPE controls the focus mode and range.

This structure can be used with [OMX\\_CONFIG\\_FOCUSREGIONTYPE](#) to specify the focus regions of interest.

OMX\_IMAGE\_CONFIG\_FOCUSCONTROLTYPE is defined as follows.

```
typedef struct OMX_IMAGE_CONFIG_FOCUSCONTROLTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_IMAGE_FOCUSCONTROLTYPE eFocusControl;
    OMX_U32 nFocusSteps;
    OMX_U32 nFocusStepIndex;
} OMX_IMAGE_CONFIG_FOCUSCONTROLTYPE;
```

##### 4.4.6.1 Parameters

The parameters for OMX\_IMAGE\_CONFIG\_FOCUSCONTROLTYPE are defined as follows.

- nPortIndex is the read only value containing the index of the portrepresents the port that this structure applies to.
- eFocusControl is an enumeration that specifies the image focus controls. [Table 4-86](#) shows the supported image focus controls.

**Table 4-86: Supported Image Focus Controls**

Field Name	Focus Control Description
OMX_IMAGE_FocusControlOn	<p>Focus control On</p> <p>Focus adjustments are being performed manually by the user.</p> <p>Focus status determination is performed by the component, <i>and status is provided via OMX_PARAM_FOCUSSTATUSTYPE (OMX_IndexConfigCommonFocusStatus)</i></p>
OMX_IMAGE_FocusControlOff	<p>Focus control off</p> <p>Focus adjustments are being performed manually by the user.</p> <p>Focus status determination is performed manually (visually inspection via viewfinder) by the user.</p>
OMX_IMAGE_FocusControlAuto	<p>Auto focus control on</p> <p>Focus adjustments are being performed automatically and continuously by the component until a capture request is issued.</p> <p>Focus status determination is performed by the component, <i>and status is provided via OMX_PARAM_FOCUSSTATUSTYPE (OMX_IndexConfigCommonFocusStatus)</i></p>
OMX_IMAGE_FocusControlAutoLock	<p>Auto focus control with lock support on</p> <p>Focus adjustment is locked to the current focus adjustment setting.</p> <p>Focus status determination is performed by the component, <i>and status is provided via OMX_PARAM_FOCUSSTATUSTYPE (OMX_IndexConfigCommonFocusStatus)</i></p> <p>The focus status request for this mode continually reflects the focus status upon receiving this lock focus request.</p>

*Note: the IL client can use OMX\_IndexConfigCommonFocusRegion to change the focus area in any of the above modes.*

- nFocusSteps is a value that specifies the number of steps that the focus can take on. The range is 0 mm to infinity.
- nFocusStepIndex defines the current position of the focus.

#### **4.4.7 OMX\_IMAGE\_PARAM\_QFACTORTYPE**

OMX\_IMAGE\_PARAM\_QFACTORTYPE determines the quality factor for JPEG compression, which controls the tradeoff between image quality and size. Q Factor provides a simpler means of controlling the JPEG compression quality than directly programming quantization tables for chroma and luma.

OMX\_IMAGE\_PARAM\_QFACTORTYPE is defined as follows.

```
typedef struct OMX_IMAGE_PARAM_QFACTORTYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_U32 nQFactor;  
} OMX_IMAGE_PARAM_QFACTORTYPE;
```

##### **4.4.7.1 Parameters**

The parameters for OMX\_IMAGE\_PARAM\_QFACTORTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the portrepresents the port that this structure applies to.
- nQFactor is a compression quality factor value in the range 1–100. A factor of 1 produces the smallest, worst quality images, and a factor of 100 produces the largest, best quality images. A typical default is 75 for small, good quality images.

#### **4.4.8 OMX\_IMAGE\_PARAM\_QUANTIZATIONTABLETYPE**

OMX\_IMAGE\_PARAM\_QUANTIZATIONTABLETYPE provides JPEG quantization tables, which are used to determine DCT compression for YUV data.

OMX\_IMAGE\_PARAM\_QUANTIZATIONTABLETYPE is an alternative to specifying Q factor, providing exact control of compression.

OMX\_IMAGE\_PARAM\_QUANTIZATIONTABLETYPE is defined as follows.

```
typedef struct OMX_IMAGE_PARAM_QUANTIZATIONTABLETYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nPortIndex;  
    OMX_IMAGE_QUANTIZATIONTABLETYPE eQuantizationTable;  
    OMX_U8 nQuantizationMatrix[64];  
} OMX_IMAGE_PARAM_QUANTIZATIONTABLETYPE;
```

#### 4.4.8.1 Parameters

The parameters for OMX\_IMAGE\_PARAM\_QUANTIZATIONTABLETYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- eQuantizationTable is an enumeration for the quantization table type, which defines luma or chroma table types. [Table 4-87](#) shows the supported image quantization table types.

**Table 4-87: Supported Image Quantization Table Types**

Field Name	Quantization Table Description
OMX_IMAGE_QuantizationTableLuma	Quantization table for the luma coefficients
OMX_IMAGE_QuantizationTableChroma	Quantization table for both the Cb and Cr chroma coefficients
OMX_IMAGE_QuantizationTableChromaCb	Quantization table for Cb chroma coefficients only
OMX_IMAGE_QuantizationTableChromaCr	Quantization table for Cr chroma coefficients only
<b>OMX_IMAGE_QuantizationTableMax</b>	<b>Max value</b>

- nQuantizationMatrix is the JPEG quantization table of coefficients stored in increasing columns and then by rows of data (i.e., row 1,... row 8). Quantization values are in the range 0–255 and are stored in linear order (i.e., the component will zigzag the quantization table data internally if required).

#### 4.4.8.2 Error Conditions

On processing the OMX\_IMAGE\_PARAM\_QUANTIZATIONTABLETYPE structure, the following error conditions can occur:

- OMX\_ErrorSeparateTablesUsed when OMX\_GetParameter function is called using OMX\_IMAGE\_QuantizationTableChroma and separate quantization tables are used for the Chroma (Cb and Cr) coefficients.

This error indicates that separate OMX\_GetParameter function calls need to be issued using OMX\_IMAGE\_QuantizationTableChromaCb and OMX\_IMAGE\_QuantizationTableChromaCr to query for the separate chroma coefficient quantization tables.

#### 4.4.9 OMX\_IMAGE\_PARAM\_HUFFMANTABLETYPE

The OMX\_IMAGE\_PARAM\_HUFFMANTABLETYPE structure is used to set the Huffman variable code length type used for JPEG.

OMX\_IMAGE\_PARAM\_HUFFMANTABLETYPE is defined as follows.

```

typedef struct OMX_IMAGE_PARAM_HUFFMANTABLETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_IMAGE_HUFFMANTABLETYPE eHuffmanTable;
    OMX_U8 nNumberOfHuffmanCodeOfLength[16];
    OMX_U8 nHuffmanTable[256];
}OMX_IMAGE_PARAM_HUFFMANTABLETYPE;

```

#### 4.4.9.1 Parameters

The parameters for OMX\_IMAGE\_PARAM\_HUFFMANTABLETYPE are defined as follows.

- nPortIndex is the read only value containing the index of the port represents the port that this structure applies to.
- eHuffmanTable is an enumeration for the Huffman table types. [Table 4-88](#) shows the supported Huffman table types.

**Table 4-88: Supported Huffman Table Types**

Field Name	Huffman Table Description
OMX_IMAGE_HuffmanTableAC	Huffman table to be applied to Luma and Chroma AC coefficients
OMX_IMAGE_HuffmanTableDC	Huffman table to be applied to Luma and Chroma DC coefficients
OMX_IMAGE_HuffmanTableACLuma	Huffman table to be applied to Luma AC coefficients only
OMX_IMAGE_HuffmanTableACChroma	Huffman table to be applied to Chroma AC coefficients only
OMX_IMAGE_HuffmanTableDCLuma	Huffman table to be applied to Luma DC coefficients only
OMX_IMAGE_HuffmanTableDCChroma	Huffman table to be applied to Chroma DC coefficients only
<b>OMX_IMAGE_HuffmanTableMax</b>	<b>Maximum value</b>

- nNumberOfHuffmanCodeOfLength is a value in the range of 0–16 that represents the number of Huffman codes of each possible length.
- nHuffmanTable is a value in the range of 0–255. The table sizes used for AC and DC Huffman tables are 16 and 162.

#### 4.4.9.2 Error Conditions

On processing the OMX\_IMAGE\_PARAM\_HUFFMANTABLETYPE structure, the following error conditions can occur:

- OMX\_ErrorSeparateTablesUsed when the OMX\_GetParameter function is called using OMX\_IMAGE\_HuffmanTableAC or

`OMX_IMAGE_HuffmanTableDC` and separate Huffman tables are used for the Luma and Chroma coefficients.

This error indicates that separate `OMX_GetParameter` function calls need to be issued using `OMX_IMAGE_HuffmanTableACLuma` and `OMX_IMAGE_HuffmanTableACChroma` to obtain the AC coefficient information and separate `OMX_GetParameter` function calls need to be issued using `OMX_IMAGE_HuffmanTableDCLuma` and `OMX_IMAGE_HuffmanTableDCChroma` to obtain the DC coefficient information.

#### **4.4.10 OMX CONFIG FLICKERREJECTIONTYPE**

`OMX_CONFIG_FLICKERREJECTIONTYPE` is used to specify the flicker rejection mode, generally used to account for the flicker effect noticeable under electric lighting.

`OMX_CONFIG_FLICKERREJECTIONTYPE` is defined as follows.

```
typedef struct OMX_CONFIG_FLICKERREJECTIONTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_FLICKERREJECTIONTYPE eFlickerRejection;
} OMX_CONFIG_FLICKERREJECTIONTYPE;
```

##### **4.4.10.1 Parameters**

The parameters for `OMX_CONFIG_FLICKERREJECTIONTYPE` are defined as follows.

- `nPortIndex` represents the port that this structure applies to.
- `eFlickerRejection` is an enumerated value indicating the flicker rejection mode.

**Table 4-89: eFlickerRejection Values**

<b><u>OMX_FLICKERREJECTIONTYPE value</u></b>	<b><u>Mode Description</u></b>
<code>OMX_FlickerRejectionOff</code>	No flicker rejection
<code>OMX_FlickerRejectionAuto</code>	Automatic flicker rejection
<code>OMX_FlickerRejection50</code>	Flicker rejection at 50Hz
<code>OMX_FlickerRejection60</code>	Flicker rejection at 60Hz

#### **4.4.11 OMX IMAGE HISTOGRAMTYPE**

The image histogram is measured on the data image input and gives the number of pixels for each tonal value. The result is delivered with the `OMX_IMAGE_HISTOGRAMDATATYPE` structure. It is possible that not all histogram types are supported by a camera, or even that a camera might not support all histogram types depending on what its port settings are. This support information is retrieved with the `OMX_IMAGE_HISTOGRAMINFOTYPE` structure.

Histogram measurements can be controlled by the following data structure.

```
typedef struct OMX_IMAGE_HISTOGRAMTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nBins;
    OMX_HISTOGRAMTYPE eHistType;
} OMX_IMAGE_HISTOGRAMTYPE;
```

#### 4.4.11.1 Parameters

The parameters for `OMX_IMAGE_HISTOGRAMTYPE` are defined as follows.

- `nPortIndex` represents the port that this structure applies to.
- `nBins` specifies the number of histogram bins. When queried with set to zero, the response gives the maximum number of bins allowed.
- `eHistType` is an enumeration specifying the histogram type, as shown in Table 4-90. This parameter is also used to enable and disable histogram generation.

Table 4-90: Histogram control type Values

<u>OMX_HISTOGRAMTYPE value</u>	<u>Histogram Description</u>
<u>OMX_Histogram_Off</u>	Histogram is disabled.
<u>OMX_Histogram_RGB</u>	RGB Color (R, G, B) histogram mode (and implicitly enables histogram if previously off).
<u>OMX_Histogram_Luma</u>	Luma (Y) histogram mode (and implicitly enables histogram if previously off).
<u>OMX_Histogram_Chroma</u>	Chroma (Cb, Cr) histogram (and implicitly enables histogram if previously off).

#### 4.4.12 OMX\_IMAGE\_HISTOGRAMDATATYPE

The histogram estimation data is described with the following structure.

```
typedef struct OMX_IMAGE_HISTOGRAMDATATYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_HISTOGRAMTYPE eHistType;
    OMX_U32 nBins;
    OMX_U8 data[1];
} OMX_IMAGE_HISTOGRAMDATATYPE;
```

#### 4.4.12.1 Parameters

The parameters for `OMX_IMAGE_HISTOGRAMDATATYPE` are defined as follows.

- nSize is the size of the structure including the length of data field containing the histogram data.
- nPortIndex represents the port that this structure applies to.
- eHistType is the read-only value specifying the histogram type, as shown in Table 4-90. Multiple histogram components may be generated depending on the type specified.
- nBins is the read-only value containing the number of bins the histogram allocates for each component.
- data[1] first byte of the histogram data. Histogram data is recorded as number of pixels per bin. For eHistType that specify multiple histogram components the size of the data will be number of histogram components times the number of bins and the data for each component will be grouped together. For example, OMX\_Histogram\_RGB has three components, thus the total number of bins recorded in data will be 3x nBins with the entire R histogram followed by entire G histogram followed by entire B histogram. The number of bits per bin may vary by histogram type and by how many bins are specified. This information can be retrieved with the OMX\_IMAGE\_HISTOGRAMINFOTYPE structure.

#### 4.4.13 OMX IMAGE HISTOGRAMINFOTYPE

This structure is used to retrieve the histogram types supported by the component. It will also report the maximum number of bins for the histogram type as well as the number of bits used to represent the histogram data per bin.

```
typedef struct OMX_IMAGE_HISTOGRAMINFOTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_HISTOGRAMTYPE eHistType;
    OMX_U32 nBins;
    OMX_U16 nBitsPerBin;
} OMX_IMAGE_HISTOGRAMINFOTYPE;
```

##### 4.4.13.1 Parameters

The parameters for OMX\_IMAGE\_HISTOGRAMDATATYPE are defined as follows.

- nSize is the size of the structure including the length of data field containing the histogram data.
- nPortIndex represents the port that this structure applies to.
- eHistType is the histogram type being queried whether it is supported or not. The types of histograms available are listed in Table 4-90.

- nBins is the read-only value containing the maximum number of bins available for the specified eHistType. If the histogram type is not supported then nBins will be 0.
- nBitsPerValue is the number of bits per bin.

#### **4.4.14 OMX CONFIG FILEFORMATTYPE**

OMX\_CONFIG\_FILEFORMATTYPE is used to specify how the component shall interpret the URI, for example if it is formatted in DCF form and whether it will auto-increment the file name index.

```
typedef struct OMX_CONFIG_FILEFORMATTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_FILEFORMATTYPE eFileFormat;
} OMX_CONFIG_FILEFORMATTYPE;
```

##### **4.4.14.1 Parameters**

The parameters for OMX\_CONFIG\_FILEFORMATTYPE are defined as follows.

- eFileFormat is file format type to let the component know how to interpret the given URI and whether any characters are to be incremented.

**4-91: FileFormat Types**

<u>OMX_FILEFORMATTYPE value</u>	<u>Mode Description</u>
<u>OMX_FileFormatNone</u>	<u>No file format naming convention is followed.</u>
<u>OMX_FileFormatDCF</u>	<p><u>DCF file naming convention. The provided URI should be DCF formatted with the first four characters of the file name any alphanumeric characters and the final four a file number greater than 0 to a maximum of 9999.</u></p> <p><u>For example,</u>  <u>“file:///DCIM/100abcde/ab4_1234.jpg” where “ab4_” is the alphanumeric portion and “1234” is the file number. The component shall increment the file number part of the name until it reaches the maximum.</u></p> <p><u>The component shall return <u>OMX_ErrorNoMore</u> if the index has reached the maximum possible value.</u></p>

#### **4.4.15 IMAGE CAPTURE START-END NOTIFICATIONS**

This functionality allows IL client to be notified of the exact moments when image capturing is active.

In case of camera devices with mechanical shutter, this corresponds to the opening and closing of the shutter.

In case of devices with rolling shutter, it corresponds to the moments of exposure starting and ending.

This support can be utilized to better synchronize the actual moments of image capture with notifications provided by IL client to the user that image capture is active.

OMX\_IndexConfigImageCaptureStarted allows the IL client to be notified of when an image to be captured by the component is exposed (capture process actually starts). OMX\_IndexConfigImageCaptureEnded allows the IL client to be notified of when an image to be captured by the camera component has ended being exposed (capture process actually ends). These notifications allow for synchronization with other functionality in the system, e.g. playback of camera capture sounds or graphics.

OMX\_IndexConfigImageCaptureStarted and OMX\_IndexConfigImageCaptureEnded are both associated with the OMX\_PARAM\_U32TYPE structure. The unsigned 32-bit value in the structure is used to count the number of image capture start and end occurrences. This counter is initialized with 0.

For example, to enable notification of the exact moment of image capture start, the IL client shall subscribe to callbacks using OMX\_IndexConfigCallbackRequest with OMX\_CONFIG\_CALLBACKREQUESTTYPE::nIndex=OMX\_IndexConfigImageCaptureStarted.

When an image exposure is initiated, the camera component issues the OMX\_Event\_IndexSettingChanged event and the IL client knows that an image is being exposed. It is not expected that the IL client will need to query the value of the counter. If the IL client intends to use the counter, it is expected that it also resets its value at appropriate moments.

Similar usage is for notifications on the exact moment of image capture end.

## 4.5 “Other” Domain

This section describes the concepts, structures, and configurations for the domain designated as “other” and moniker distinguishing it from the audio, video and image domains. The OMX\_Other.h header specifies the parameters and configurations in detail.

Presently the other domain formalizes only a “time” data format and its associated operation though other data formats may be formalized in the future. The time data format exists to facilitate synchronization. To provide context to the definition of the time data format, the following section explains OpenMAX IL’s synchronization mechanisms.

### 4.5.1 Parameters and Config Indexes

The header OMX\_Index.h contains the enumeration OMX\_INDEXTYPE, which contains all of the standard index values used with the functions OMX\_GetParameter, OMX\_SetParameter, OMX\_GetConfig, and OMX\_SetConfig. [Table 4-92](#) describes the index values that relate to Other Domain.

**Table 4-92: Index Values for Other Domain**

Index	Description
OMX_IndexConfigTimeScale	Used with OMX_GetConfig and OMX_SetConfig to access a OMX_TIME_CONFIG_SCALETYPE structure denoting the scale of the media clock.
OMX_IndexConfigTimeClockState	Used with OMX_GetConfig and OMX_SetConfig to access a OMX_TIME_CONFIG_CLOCKSTATETYPE structure denoting the state of the media clock.
<del>OMX_IndexConfigTimeActiveRefClock</del>	<del>Used with OMX_GetConfig and OMX_SetConfig to access a OMX_TIME_CONFIG_ACTIVEREFCLOCKTYPE structure denoting the active reference clock.</del>
OMX_IndexConfigTimeCurrentMediaTime	Used with OMX_GetConfig to query a OMX_TIME_CONFIG_TIMESTAMPTYPE structure denoting the current media time.
OMX_IndexConfigTimeCurrentWallTime	Used with OMX_GetConfig to query a OMX_TIME_CONFIG_TIMESTAMPTYPE structure denoting the current wall clock time.

Index	Description
<code>OMX_IndexConfigTimeCurrentAudioReference</code>	<u>Used with OMX_SetConfig to set the OMX_TIME_CONFIG_TIMESTAMPTYPE structure denoting the current audio reference clock time time.</u>
<code>OMX_IndexConfigTimeCurrentVideoReference</code>	<u>Used with OMX_SetConfig to set the OMX_TIME_CONFIG_TIMESTAMPTYPE structure denoting the current video reference clock time time.</u>
<code>OMX_IndexConfigTimeMediaTimeRequest</code>	Used with OMX_SetConfig to request a clock component operation using a OMX_TIME_CONFIG_MEDIATIMEREQUESTTYPE structure.
<code>OMX_IndexConfigTimeClientStartTime</code>	Used with OMX_SetConfig to set the start time of the given client stream using the OMX_TIME_CONFIG_TIMESTAMPTYPE structure.
<code>OMX_IndexConfigTimePosition</code>	Used with OMX_GetConfig and OMX_SetConfig to access a OMX_TIME_CONFIG_SCALETYPE structure denoting the current position in time.
<code>OMX_IndexConfigTimeSeekMode</code>	Used with OMX_GetConfig and OMX_SetConfig to access a OMX_TIME_CONFIG_SCALETYPE structure denoting the current seek mode.
<code>OMX_IndexConfigTimeCurrentReference</code>	<u>Used with OMX_SetConfig to set the OMX_TIME_CONFIG_TIMESTAMPTYPE structure denoting the current reference clock time.</u>
<code>OMX_IndexConfigTimeActiveRefClockUpdate</code>	<u>Used with OMX_SetConfig to set the OMX_TIME_CONFIG_ACTIVEREFCLKUPDATETYPE structure denoting the role of reference clock provider to clock clients.</u>
<code>OMX_IndexConfigTimeUpdate</code>	<u>Used with OMX_SetConfig to pass updates from the clock to its clients using the OMX_TIME_MEDIATIMETYPE structure.</u>

#### 4.5.2 OMX\_TIME\_CONFIG\_SEEKMODETYPE

A component's seek mode defines the semantics it follows when an IL client requests a change in position (via the `OMX_IndexConfigTimePosition` configuration ).

`OMX_TIME_CONFIG_SEEKMODETYPE` is defined as follows.

```

typedef struct OMX_TIME_CONFIG_SEEKMODETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_TIME_SEEKMODETYPE eType;
} OMX_TIME_CONFIG_SEEKMODETYPE;

```

#### 4.5.2.1 Parameters

The parameters for OMX\_TIME\_CONFIG\_SEEKMODETYPE are defined as follows.

- eType is seek mode and must be a value from the OMX\_TIME\_SEEKMODETYPE enumeration

**Table 4-93: Seek Modes Defined by OMX\_TIME\_SEEKMODETYPE**

Field Name	Description
OMX_TIME_SeekModeFast	Prefer seeking to an approximation of the requested seek position over the actual seek position if it results in a faster seek.
OMX_TIME_SeekModeAccurate	Prefer seeking to the actual seek position over an approximation of the requested seek position even if it results in a slower seek.

#### 4.5.3 OMX\_TIME\_CONFIG\_TIMESTAMPTYPE

A timestamp represents a position in time relative to some clock. The OMX\_IndexConfigTimeCurrentWallTime, OMX\_IndexConfigTimeCurrentMediaTime, ~~OMX\_IndexConfigTimeCurrentAudioReference~~, and OMX\_IndexConfigTimeCurrentVideoReference configurations leverage this structure.

OMX\_TIME\_CONFIG\_TIMESTAMPTYPE is defined as follows.

```

typedef struct OMX_TIME_CONFIG_TIMESTAMPTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_TICKS nTimestamp;
} OMX_TIME_CONFIG_TIMESTAMPTYPE;

```

#### 4.5.3.1 Parameters

The parameters for OMX\_TIME\_CONFIG\_TIMESTAMPTYPE are defined as follows.

- nPortIndex is the read-only value containing the index of the port represents the port that this structure applies to.
- nTimestampType holds the actual timestamp value.

#### **4.5.4    OMX\_TIME\_CONFIG\_MEDIATIMEREQUESTTYPE**

The media time request represents a request for notification at the media time specified.

OMX\_TIME\_CONFIG\_MEDIATIMEREQUESTTYPE is defined as follows.

```
typedef struct OMX_TIME_CONFIG_MEDIATIMEREQUESTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_PTR pClientPrivate;
    OMX_TICKS nMediaTimestamp;
    OMX_TICKS nOffset;
} OMX_TIME_CONFIG_MEDIATIMEREQUESTTYPE;
```

##### **4.5.4.1    Parameters**

The parameters for OMX\_TIME\_CONFIG\_MEDIATIMEREQUESTTYPE are defined as follows.

- nPortIndex is the read only value containing the index of the portrepresents the port that this structure applies to.
- pClientPrivate client private data to disambiguate this media time from others.
- nMediaTimestamp media time requested.
- nOffset amount of wall clock time by which this request should be fulfilled early.

#### **4.5.5    OMX\_TIME\_MEDIATIMETYPE**

The media time structure is sent to a port either to fulfill a media time request or when the clock state or scale has changed. This structure is either used with the index OMX\_IndexConfigTimeUpdate and a call to OMX\_SetConfig if the port is tunneled, or written into the payload of a buffer and sent via the OMX\_FillBufferDone callback to the client.

OMX\_TIME\_MEDIATIMETYPE is defined as follows.

```

typedef struct OMX_TIME_MEDIATIMETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nClientPrivate;
    OMX_TIME_UPDATETYPE eUpdateType;
    OMX_TICKS nMediaTimestamp;
    OMX_TICKS nOffset;
    OMX_TICKS nWallTimeAtMediaTime;
    OMX_S32 xScale;
    OMX_TIME_CLOCKSTATE eState;
} OMX_TIME_MEDIATIMETYPE;

```

#### 4.5.5.1 Parameters

The parameters for OMX\_TIME\_MEDIATIMETYPE are defined as follows.

- pClientPrivate clock client private data to disambiguate this media time from others. If the eUpdateType field indicates this is scale or state change notification, the pClientPrivate field shall be zero.
- eUpdateType designates reason for the this update was sent and must be a value from the OMX\_TIME\_UPDATETYPE enumeration

**Table 4-94: Media Time Update Types Defined by OMX\_TIME\_UPDATETYPE**

Field Name	Description
OMX_TIME_UpdateRequestFulfillment	Update is the fulfillment of a media time request.
OMX_TIME_UpdateScaleChanged	Update to indicate the clock scale has changed.
OMX_TIME_UpdateClockStateChanged	Update to indicate the clock state has changed.

- nMediaTimeStamp denotes the media time requested (if this is a request fulfillment).
- nOffset designates amount of wall clock time by which this request was actually fulfilled early (if this is a request fulfillment).
- nWallTimeAtMediaTime denotes the wall time corresponding to nMediaTimeStamp (if this is a request fulfillment).
- xScale designates the current media time scale in Q16 format when the structure was completed.
- eState designates the clock state when the structure was completed, and must be a value from the OMX\_TIME\_CLOCKSTATE enumeration

**Table 4-95: Clock States Defined by OMX\_TIME\_CLOCKSTATE**

Field Name	Description
OMX_TIME_ClockStateRunning	Clock is running.
OMX_TIME_ClockStateWaitingForStartTime	Clock is waiting until the prescribed clients emit their start time.
OMX_TIME_ClockStateStopped	Clock is stopped.

#### **4.5.6 OMX\_TIME\_CONFIG\_SCALETYPE**

The clock scale config represents the current clock scale. It allows the IL client to query and set the clock scale.

OMX\_TIME\_CONFIG\_SCALETYPE is defined as follows.

```
typedef struct OMX_TIME_CONFIG_SCALETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_S32 xScale;
} OMX_TIME_CONFIG_SCALETYPE;
```

##### **4.5.6.1 Parameters**

The parameters for OMX\_TIME\_CONFIG\_SCALETYPE are defined as follows.

- xScale the scale of the media time in Q16 format.

#### **4.5.7 OMX\_TIME\_CONFIG\_CLOCKSTATETYPE**

The clock state config represents the current state of the media clock. It allows the IL client to set and query the clock state.

OMX\_TIME\_CONFIG\_CLOCKSTATETYPE is defined as follows.

```
typedef struct OMX_TIME_CONFIG_CLOCKSTATETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_TIME_CLOCKSTATE eState;
    OMX_TICKS nStartTime;
    OMX_TICKS nOffset;
    OMX_U32 nWaitMask;
} OMX_TIME_CONFIG_CLOCKSTATETYPE;
```

##### **4.5.7.1 Parameters**

The parameters for OMX\_TIME\_CONFIG\_CLOCKSTATETYPE are defined as follows.

- `eState` denotes the state of the media clock and must be a value in the `OMX_TIME_CLOCKSTATE` enumeration.
- `nStartTime` designates the media time the media clock is initialized to.
- `nOffset` designates the time to offset the media time by.
- `nOffset` specifies a mask of `OMX_CLOCKPORT` values designating which ports, if any, to wait on.

**Table 4-96: Possible Clock Port Values**

Field Name	Value
<code>OMX_CLOCKPORT0</code>	0x00000001
<code>OMX_CLOCKPORT1</code>	0x00000002
<code>OMX_CLOCKPORT2</code>	0x00000004
<code>OMX_CLOCKPORT3</code>	0x00000008
<code>OMX_CLOCKPORT4</code>	0x00000010
<code>OMX_CLOCKPORT5</code>	0x00000020
<code>OMX_CLOCKPORT6</code>	0x00000040
<code>OMX_CLOCKPORT7</code>	0x00000080

#### **4.5.8 OMX\_TIME\_CONFIG\_ACTIVEREFCLOCKTYPE**

The active reference clock structure represents the clock currently being used as a reference for the media clock. It allows the IL client to set and query the currently active reference clock.

`OMX_TIME_CONFIG_ACTIVEREFCLOCKTYPE` is defined as follows:

```
typedef struct OMX_TIME_CONFIG_ACTIVEREFCLOCKTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_TIME_REFCLKTYPE eClock;
} OMX_TIME_CONFIG_ACTIVEREFCLOCKTYPE;
```

##### **4.5.8.1 Parameters**

The parameters for `OMX_TIME_CONFIG_ACTIVEREFCLOCKTYPE` are defined as follows:

- `eClock` denotes the currently active reference clock and must be a value in the `OMX_TIME_REFCLKTYPE` enumeration.

**Table 4-76: Reference Clock Enumeration**

Field Name	Value
<code>OMX_TIME_RefClockNone</code>	No active reference clock.
<code>OMX_TIME_RefClockAudio</code>	The audio clock is the active reference clock.
<code>OMX_TIME_RefClockVideo</code>	The video clock is the active reference clock.

## 4.6 Common or Domain Independent

This section describes the parameter and configuration details for controls that are either domain independent or applicable for all domains.

### 4.6.1 Parameter and Configuration Indices

The header OMX\_Index.h contains the enumeration `OMX_INDEXTYPE`, which contains all standard index values used core functions `OMX_GetParameter`, `OMX_SetParameter`, `OMX_GetConfig`, and `OMX_SetConfig`. Table 4-97 shows the index values that relate to domain independent support.

**Table 4-97: Domain Independent Indices**

OpenMAX IL Indices (OMX_Index.h)	Corresponding OpenMAX IL Structures
<code>OMX_IndexConfigCommitMode</code>	<code>OMX_CONFIG_COMMITMODETYPE</code>
<code>OMX_IndexConfigCommit</code>	<code>OMX_CONFIG_COMMITTYPET</code>
<code>OMX_IndexConfigCallbackRequest</code>	<code>OMX_CONFIG_CALLBACKREQUESTTYPE</code>
<code>OMX_IndexParamMediaContainer</code>	<code>OMX_MEDIACONTAINER_INFOTYPE</code>
<code>OMX_IndexParamReadOnlyBuffers</code>	<p><code>OMX_PORTBOOLEANTYPE</code></p> <p>Used for setting and querying the use of ready-only buffers, i.e., buffers marked with <code>OMX_BUFFERFLAG_READONLY</code>. By querying this index on an output port, the IL client can determine whether the output port will produce read-only buffers. By enabling this setting on an input port, the IL client can notify the component on its intention to send read-only buffers to the port. Setting this parameter on an output port is not supported.</p>

For example, `OMX_IndexConfigCommitMode` index is used with `OMX_GetConfig` and `OMX_SetConfig` to access `OMX_CONFIG_COMMITMODETYPE`.

#### **4.6.2 OMX\_CONFIG\_COMMITMODETYPE**

The OMX\_CONFIG\_COMMITMODETYPE structure is used for configuring the component to operate in a deferred or immediate mode.

In deferred mode, all settings (OMX\_SetConfig() calls) are cached and not applied until the IL client issues an explicit commit request (OMX\_IndexConfigCommit). Upon receiving the commit request, the component shall apply all the cached setting atomically.

In immediate mode, the component does not cache any settings and applies the settings immediately. Immediate mode shall be the default mode of operation.

In the case, a component is requested to transition from deferred to immediate mode while holding onto cached settings (OMX\_IndexConfigCommit has not been issued), the component shall discard all the cached setting requests and complete the transition to immediate mode without any changes to its configuration settings.

At the time of commit, all cached settings are to be applied simultaneously and atomically; the order of the individual OMX\_SetConfig() calls shall not affect the end result. However, if the IL client issues the same setting request multiple times before commit, the last setting request shall override the earlier settings.

If the IL client calls OMX\_GetConfig() on a setting that is currently cached, the result shall reflect the current active value and not the setting request currently cached.

Requesting the component to transition to deferred mode while the component is currently configured in deferred mode shall result in an error (OMX\_ErrorInvalidMode).

OMX\_CONFIG\_COMMITMODETYPE is defined as follows.

```
typedef struct OMX_CONFIG_COMMITMODETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_BOOL bDeferred;
} OMX_CONFIG_COMMITMODETYPE;
```

##### **4.6.2.1 Parameters**

The parameters for OMX\_CONFIG\_COMMITMODETYPE are defined as follows.

- eDeferred specifies the mode of operation. OMX\_TRUE shall indicate deferred mode and OMX\_FALSE shall indicate immediate mode.  
A component shall default to immediate mode (OMX\_FALSE).

##### **4.6.2.2 Error Conditions**

On processing the OMX\_CONFIG\_COMMITMODETYPE structure, the following error conditions can occur:

- OMX\_ErrorInvalidMode when the component is being requested to transition to deferred mode while currently configured for deferred mode.

### **4.6.3 OMX CONFIG COMMITTYPE**

The OMX\_CONFIG\_COMMITTYPE structure is used to commit previously cached settings.

This functionality is only valid if the component is configured for deferred mode of operation, otherwise an error shall be issued.

OMX\_CONFIG\_COMMITTYPE is defined as follows.

```
typedef struct OMX_CONFIG_COMMITTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
} OMX_CONFIG_COMMITTYPE;
```

#### **4.6.3.1 Parameters**

None

#### **4.6.3.2 Functionality**

If an OMX\_SetConfig() fails (returns an error code) while a component is in the deferred mode, the next commit request shall fail with OMX\_ErrorBadParameter. For these cases when OMX\_SetConfig() fails, it is recommended that an IL client immediately issues a commit in order to flush the cached settings.

It is also possible that separate configs have interdependencies, e.g., a particular setting of one config may restrict the valid value range of another setting. The component is not required to immediately validate the current OMX\_SetConfig() settings, it is however required to perform this validation at the time of commitment. In such cases the individual OMX\_SetConfig() calls can fail with an error code only if the component will not support a particular setting in any situation. If all the individual OMX\_SetConfig() calls have succeeded, but the component determines the combination invalid, it shall signal the IL client by returning OMX\_ErrorBadParameter from the commit OMX\_SetConfig() call. When a commit fails, all settings cached in the component shall be discarded.

#### **4.6.3.3 Error Conditions**

On processing the OMX\_CONFIG\_COMMITTYPE structure, the following error conditions can occur:

- OMX\_ErrorInvalidMode when the component is configured for immediate mode of operation (refer to OMX\_CONFIG\_COMMITMODETYPE).
- OMX\_ErrorBadParameter when a OMX\_SetConfig() cache settings call failed
- OMX\_ErrorBadParameter when the component determines the combinational cached settings are invalid.

#### **4.6.4 OMX\_CONFIG\_CALLBACKREQUESTTYPE**

The OMX\_CONFIG\_CALLBACKREQUESTTYPE structure is used to signal setting changes associated with a parameter or config index (OMX\_INDEXTYPE).

The notification is associated with the OMX\_EventIndexSettingChanged event, the event callback includes the parameter or config index that is associated with the setting change. When receiving the event, the IL client shall use OMX\_GetParameter() or OMX\_GetConfig() as appropriate to retrieve the new value of the parameter or config.

The callback settings are fully independent of any other settings applied to the component, including component state.

OMX\_CONFIG\_CALLBACKREQUESTTYPE is defined as follows.

```
typedef struct OMX_CONFIG_CALLBACKREQUESTTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_INDEXTYPE nIndex;
    OMX_BOOL bEnabled;
} OMX_CONFIG_CALLBACKREQUESTTYPE;
```

##### **4.6.4.1 Parameters**

The parameters for OMX\_CONFIG\_CALLBACKREQUESTTYPE are defined as follows.

- nPortIndex is the value containing the index of the port (can be OMX\_ALL).
- nIndex specifies the OMX\_INDEXTYPE index to be associated with the event notification.
- bEnabled is a Boolean field that indicates if event notification for the nIndex shall be enabled.

By default event notifications associated with this functionality are disabled.

#### **4.6.5 OMX\_MEDIACONTAINER\_INFOTYPE**

The OMX\_MEDIACONTAINER\_INFOTYPE structure identifies the media container format.

OMX\_MEDIACONTAINER\_INFOTYPE is defined as follows.

```
typedef struct OMX_MEDIACONTAINER_INFOTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_MEDIACONTAINER_FORMATTYPE eFmtType;
} OMX_MEDIACONTAINER_INFOTYPE;
```

#### 4.6.5.1 Parameters

The parameters for OMX\_MEDIACONTAINER\_INFOTYPE are defined as follows.

- eFmtType specifies the media container format type.

Table 4-98 : Media Container Formats

<u>OMX_MEDIACONTAINER_FORMATTYPE</u> Enumerated Value	<u>Description</u>
<u>OMX_FORMAT_RAW</u>	<u>No Format</u>
<u>OMX_FORMAT_MP4</u>	<u>Mpeg4 File</u>
<u>OMX_FORMAT_3GP</u>	<u>3GP File</u>
<u>OMX_FORMAT_3G2</u>	<u>3G2 File</u>
<u>OMX_FORMAT_AMC</u>	<u>AMC file</u>
<u>OMX_FORMAT_SKM</u>	<u>SKM file</u>
<u>OMX_FORMAT_K3G</u>	<u>K3G file</u>
<u>OMX_FORMAT_VOB</u>	<u>VOB file</u>
<u>OMX_FORMAT_AVI</u>	<u>AVI File</u>
<u>OMX_FORMAT ASF</u>	<u>ASF File</u>
<u>OMX_FORMAT_RM</u>	<u>Real Media</u>
<u>OMX_FORMAT_MPEG_ES</u>	<u>Mpeg2 ES</u>
<u>OMX_FORMAT_DIVX</u>	<u>Divx file</u>
<u>OMX_FORMAT_MPEG_TS</u>	<u>Mpeg2 TS</u>
<u>OMX_FORMAT_QT</u>	<u>Quicktime</u>
<u>OMX_FORMAT_M4A</u>	<u>M4A file</u>
<u>OMX_FORMAT_MP3</u>	<u>Mp3 file</u>
<u>OMX_FORMAT_WAVE</u>	<u>Wave file</u>
<u>OMX_FORMAT_XMF</u>	<u>XMF file</u>
<u>OMX_FORMAT_AMR</u>	<u>AMR file</u>
<u>OMX_FORMAT_AAC</u>	<u>AAC file</u>
<u>OMX_FORMAT_EVRC</u>	<u>EVRC file</u>
<u>OMX_FORMAT_QCP</u>	<u>QCP file</u>
<u>OMX_FORMAT_SMF</u>	<u>SMF file</u>
<u>OMX_FORMAT_OGG</u>	<u>OGG file</u>
<u>OMX_FORMAT_BMP</u>	<u>BMP file</u>

<u>OMX_FORMAT_JPG</u>	<u>JPG file</u>
<u>OMX_FORMAT_JPG2000</u>	<u>JPG2000 file</u>
<u>OMX_FORMAT_MKV</u>	<u>MKV file</u>
<u>OMX_FORMAT_FLV</u>	<u>FLV file</u>
<u>OMX_FORMAT_M4V</u>	<u>M4V file</u>
<u>OMX_FORMAT_F4V</u>	<u>F4V file</u>
<u>OMX_FORMAT_WEBM</u>	<u>WebM file</u>
<u>OMX_FORMAT_WEBP</u>	<u>WebP file</u>

#### **4.6.6 OMX CONFIG PORTBOOLEANTYPE**

OMX\_CONFIG\_PORTBOOLEANTYPE is used to specify a port specific Boolean property.

OMX\_CONFIG\_PORTBOOLEANTYPE is defined as follows.

```
typedef struct OMX_CONFIG_PORTBOOLEANTYPE{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnabled;
} OMX_CONFIG_PORTBOOLEANTYPE;
```

##### **4.6.6.1 Parameters**

The parameters for OMX\_CONFIG\_PORTBOOLEANTYPE are defined as follows.

- nPortIndex represents the port that this structure applies to.
- bEnabled is a Boolean field used to enable a port specific functionality.

## 5 OpenMAX IL Component Extension APIs

### 5.1 Description of the Extension Process

An OpenMAX IL component may support any setting defined in the OpenMAX IL specification. Vendors can add to the list of parameters and configurations not included in the standard header files. These additions are referred to as *extensions*.

Any extensions approved by Khronos are considered OpenMAX IL extensions. Any extensions not approved by Khronos are vendor-defined extensions.

OpenMAX IL extensions are defined in a predefined set of extension header files, namely:

- OMX\_CoreExt.h: OpenMAX IL core extension API
- OMX\_ComponentExt.h: OpenMAX IL component extension API
- OMX\_AudioExt.h: OpenMAX IL audio domain extension data structures
- OMX\_IVCommonExt.h: OpenMAX IL extension structures common to image and video domains
- OMX\_VideoExt.h: OpenMAX IL video domain extension data structures
- OMX\_ImageExt.h: OpenMAX IL image domain extension data structures
- OMX\_OtherExt.h : OpenMAX IL other domain extension data structures (includes A/V synchronization extensions)
- OMX\_IndexExt.h: Index of all OpenMAX IL extension data structures
- ~~OMX\_ContentPipeExt.h: Content pipe defined extensions~~

Any vendor that develops OpenMAX IL components may add to the list of standard indexes a collection of one or more custom parameters or configuration indexes. Each vendor-specific index shall have a value greater than the value of OMX\_IndexVendorStartUnused and less than the value of OMX\_IndexMax - 1. Each OpenMAX IL extension index has a value greater than the value of OMX\_IndexKhronosExtension and less than the value of OMX\_IndexVendorStartUnused - 1.

Each extension parameter or configuration index may apply to one of the four existing domains, namely audio, video, image, and “other”. It may also apply to a parameter or configuration that does not belong to any known domain.

A vendor-specific extension index to a parameter or configuration may be defined by a string and be reported in the component description documentation. The IL client may obtain the index related to this property using the component function OMX\_GetExtensionIndex. This function provides a numeric index from a string

that names the custom index. The function is specific to a component, so a component handle shall be passed to the function. The function is described in section 3.2.2.12.

The numeric index can be used with the functions `OMX_GetParameter` and `OMX_SetParameter` if the index regards a parameter or with the functions `OMX_GetConfig` and `OMX_SetConfig` if the index is a configuration index. The nature of the parameter or configuration value should be documented in the extension section of the component documentation. Khronos, or its designee, will maintain a publicly-accessible registry of OpenMAX IL extensions. These extensions are baselined to a version of an OpenMAX IL specification and may be promoted to a subsequent release of the OpenMAX IL specification.

### 5.1.1 **GetExtensionIndex**

The `OMX_GetExtensionIndex` method will translate a vendor-specific configuration or parameter string into an OpenMAX IL structure index. There is no requirement for the component to support this command for the indexes already found in the `OMX_INDEXTYPE` enumeration or in the anonymous enumeration in `OMX_IndexExt.h`, thus reducing a component's memory footprint. The component may support all vendor-supplied extension indexes not found in the `OMX_INDEXTYPE` enumeration ~~that it supports~~. This is a blocking call. The component should return from this call within five milliseconds.

The parameters for the `OMX_GetExtensionIndex` method are defined as follows.

Parameter	Description
<code>hComponent</code> [in]	The handle of the component to be accessed. This component handle is returned by the call to the <code>GetHandle</code> function.
<code>cParameterName</code> [in]	The string that the component will translate into a 32-bit index. <code>OMX_STRING</code> shall be less than 128 characters long including the trailing null byte.
<code>pIndexType</code> [out]	A pointer to <code>OMX_INDEXTYPE</code> that receives the index value.

#### 5.1.1.1 Prerequisites for This Method

This macro ~~has no prerequisites~~ can be invoked when the component is in any state except the `OMX_StateInvalid` state.

#### 5.1.1.2 Method Implementation

The following code defines the method implementation.

```
OMX_ERRORTYPE (*GetExtensionIndex)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_IN OMX_STRING cParameterName,  
    OMX_OUT OMX_INDEXTYPE* pIndexType);
```

### **5.1.2    *Custom Data Structures***

Each index refers to a structure or a memory area that stores the data for the parameter or configuration. The vendor shall provide a data container that is a vendor-specific structure within a vendor-specific header file. Khronos shall provide a data container that is an OpenMAX IL extensions structure within one of the OpenMAX IL extension header files. The header file is to be included by the component that implements the extension feature, and by the IL client that uses the extension feature.

If the data container is simply a pointer to a memory area, the IL client shall know how to manage the data. Each extension parameter shall be described in the component description document and follows the convention of standard OpenMAX IL data structures.

Each vendor-specific feature shall be documented in the component specifications, which describe the relationship between the string that defines a property, which is used with the `GetExtensionIndex` function, and the related data structure that corresponds to the index returned from `GetExtensionIndex` for the string.

### **5.1.3    *Enumerations***

OpenMAX IL enumeration types, as specified in the standard OpenMAX IL header files, may be extended using anonymous enum declarations in the OpenMAX IL extension or vendor-specific header files.

Each OpenMAX IL extension enumeration has a value greater than `OMX_<enum>KhronosExtensions` and smaller than `OMX_<enum>VendorStartUnused - 1`. Each Vendor specific extension enumeration has a value greater than `OMX_<enum>VendorStartUnused` and smaller than `OMX_<enum>Max`.

It may be necessary to cast the anonymous enum values to the standard OpenMAX IL enumeration types explicitly to avoid compilation errors.

### **5.1.4    *Promoting extensions to specification***

Extensions may be promoted to the OpenMAX-IL specification in subsequent releases of the OpenMAX-IL interface.

After promotion, the standard OpenMAX-IL header shall include a new standard enumeration value, as well as the extended enumeration value that remains in the OpenMAX IL extension file. It may be that both enumeration values point to the same feature.

## **5.2       *Examples of Using Extension Querying API***

This section shows sample code for extension APIs.

### 5.2.1 Sample Code Showing Calling Sequence

The following sample code shows an example of calling an extension API.

```
/* Set the vendor-specific filename parameter
   on a reader */
OMX_U32 eIndexParamFilename;
OMX_PTR eFileName;

OMX_GetExtensionIndex(
    hFileReaderComp,
    "OMX.CompanyXYZ.index.param.filename",
    &eIndexParamFilename);
OMX_SetParameter(hComp, eIndexParamFilename, &eFileName);
```

This following code sample shows how to use a vendor-specific parameter. **The code passes a file name to a component. The file name string does not belong to any OpenMAX IL domain; it is used only for this example.**

```
/* Get the vendor-specific mp3 faster
   decoding feature settings */
OMX_U32 eIndexParamFasterDecomp;
OMX_CUSTOM_AUDIO_STRUCTURE oFasterDecompParams;

InitializeAudioStructure(&oFasterDecompParams);

OMX_GetExtensionIndex(
    hMp3DecoderComp,
    "OMX.CompanyXYZ.index.param.fasterdecomp",
    &eIndexParamFasterDecomp);
OMX_GetParameter(hMp3DecoderComp, eIndexParamFasterDecomp,
    &oFasterDecompParams);
```

In this **second** example, a special parameter of an MP3 decoder is presented. The index eIndexParamFasterDecomp is retrieved, and the related data structure is stored in the oFasterDecompParams structure by the GetParameter function.

## 6 Synchronization

This section specifies synchronization functionality including seeking and clock component behavior.

### 6.1 Seeking Component

A component may be designated as a *seeking component* if it can change and report on its position in the data stream that it is processing. For instance, an IL client may command a seeking source component that retrieves an audio/video stream from a repository (for example, a local or remote file) to begin emitting data from a different location in the audio/video stream. Furthermore, an IL client may query the position that the source is currently emitting.

#### 6.1.1 Seeking Configurations

A seeking component shall support the following configurations:

- OMX\_IndexConfigTimePosition, which passes OMX\_TIME\_CONFIG\_TIMESTAMPTYPE as a parameter. OMX\_GetConfig returns the timestamp of the data that the component is currently emitting. OMX\_SetConfig commands the component to seek the given timestamp.
- OMX\_IndexConfigTimeSeekMode, which defines the manner in which the seek component performs the seek. Table 6-1 shows the seek modes.

**Table 6-1: Seek Modes**

Seek Mode	Interpretation
OMX_TIME_SeekModeFast	Prefers seeking an approximation of the requested seek position over the actual seek position if it results in a faster seek.
OMX_TIME_SeekModeAccurate	Prefers seeking to the requested seek position over an approximation of the requested seek position even if it results in a slower seek.

An arbitrary seek in a stream may request a target position whose data depends on data that precedes it. For example, consider the case where an IL client requests seeking an interframe in a video stream. Some amount of data prior to the target interframe shall be decoded to reconstruct the target frame starting with the first intraframe preceding the target. If fast mode is set, the seeking component may use the intraframe as an approximation of the target and start displaying frames immediately at that intraframe. If accurate mode is set, the seeking component decodes frames starting with the intraframe but does not display frames until the target position.

### **6.1.2 Seeking Buffer Flags**

A seeking component communicates the role of certain buffers in the context of seeking to its downstream components via special buffer flags. A buffer flag corresponds to the first new logical data unit in a buffer, which is the first unit with its starting boundary occurring in the buffer.

The special buffer flags of note are as follows.

- **OMX\_BUFFERFLAG\_DECODEONLY:** The seeking component sets this flag on a buffer if the buffer shall be decoded but not displayed. In the example above, if the seeking component is in accurate mode, it would set this flag on all frames preceding the target interframe. A decoder component decodes but does not propagate downstream a buffer marked decode only. A component that renders data shall ignore any buffer with this flag set.
- **OMX\_BUFFERFLAG\_STARTTIME:** The seeking component sets this flag on the buffer that carries the starting timestamp of the data stream. In the example above, the seeking component would set this flag on the intraframe (i.e., the approximation) when in fast seek mode and on the interframe (i.e., the original target) when in accurate seek mode. When a clock component client receives a buffer with this flag set, it performs an `OMX_SetConfig` call with `OMX_IndexConfigTimeClientStartTime` on the clock component that is sending the buffer's timestamp. The transmission of the start time informs the clock component that the client's stream is ready for presentation and the timestamp of the first data to be presented.

### **6.1.3 Seek Event Sequence**

To implement a seek on a chain of components, an IL client shall perform the following operations in order:

1. Pause the component through the use of `OMX_SendCommand` requesting a state transition to `OMX_StatePause`.
2. Stop the clock component's media clock through the use of `OMX_SetConfig` on `OMX_TIME_CONFIG_CLOCKSTATETYPE` requesting a transition to `OMX_TIME_ClockStateStopped`.
3. Seek to the desired location through the use of `OMX_SetConfig` on `OMX_IndexConfigTimePosition` requesting the desired timestamp.
4. Flush all components.
5. Start the clock component's media clock through the use of `OMX_SetConfig` on `OMX_TIME_CONFIG_CLOCKSTATETYPE` requesting a transition to either `OMX_TIME_ClockStateRunning` or `OMX_TIME_ClockStateWaitingForStartTime`.
6. Un-pause the component through the use of `OMX_SendCommand` requesting a state transition to `OMX_StateExecuting`.

If the IL client requests a transition to OMX\_TIME\_ClockStateRunning, the clock component immediately starts the media clock using the designated start time. This is a simpler transition than going to OMX\_TIME\_ClockStateWaitingForStartTime but may compromise synchronization at the start of playback after a seek operation since it ignores the start times of the individual media streams.

If the IL client requests a transition to OMX\_TIME\_ClockStateWaitingForStartTime, it designates which clock component clients to wait for. The clock component then waits for these clients to send their start times via the OMX\_IndexConfigTimeClientStartTime configuration. Once all required clients have responded, the clock component starts the media clock using the earliest client start time. This approach ensures the following:

- All clients are ready to render data, eliminating any initial drift between streams.
- The media clock start time reflects the clocks of all clients and any adjustment made by the seeking component.

## 6.2 Clock Component

OpenMAX IL defines a special component denoted the *clock component* to facilitate the smooth and synchronized delivery or capture of audio and video streams as well as rate control. The clock component takes ~~one audio and one video~~ reference clock as input, from which it derives a media clock. The clock component shares the media time with the clients with whom it is connected via clock ports (one clock port per client). The clock component also exposes a mechanism for controlling the media clock and makes clients aware of the rate control events via their clock ports.

### 6.2.1 Timestamps

All timestamps and durations are expressed as OMX\_TICKS values as shown in the following structure.

```
typedef struct OMX_TICKS
{
    OMX_U32 nLowPart;
    OMX_U32 nHighPart;
} OMX_TICKS;
```

This structure shall be interpreted as a signed 64-bit value representing microseconds. This representation accommodates the following:

- Positive and negative time values. Examples of negative time values include pre-roll timestamp and time deltas.
- High-resolution timestamps (e.g., MPEG2 presentation timestamps based on a 90 kHz clock) and allow more accurate and synchronized delivery (e.g., individual audio samples delivered at 192 kHz).
- A large dynamic range of approximately plus or minus 26 million days; 32-bit resolution provides a range of only about plus or minus 35 minutes.

Implementations with limited precision may convert the signed 64-bit value to a signed 32-bit value internally but risk loss of precision.

### 6.2.2 Media Clock

The clock component maintains a media clock that tracks the current position in the media stream. The instantaneous media time is represented as the timestamp, relative to the start of the stream, of the data being delivered or captured at that instant (e.g., the current audio sample). Consequently, media time increases (corresponding to playing or fast forwarding), decreases (corresponding to rewinding), or holds at some constant (corresponding to pausing) according to the rate control applied to the media clock.

The clock component can be queried for the current media clock time using `OMX_GetConfig` with the read-only index `OMX_IndexConfigTimeCurrentMediaTime` and structure `OMX_TIME_CONFIG_TIMESTAMPTYPE`.

The current media clock time is written into the `nTimestamp` field. This index must be used with the `nPortIndex` field as `OMX_ALL`, since the media clock is not specific to any port.

#### 6.2.2.1 Media Clock Scale

The clock component maintains the media time's current scale factor, which corresponds directly to the rate control applied on it. The scale is a Q16 value relative to a 1X forward advancement of the media clock. Thus, scale ranges map to modes of playback, as shown in Figure 6-1.

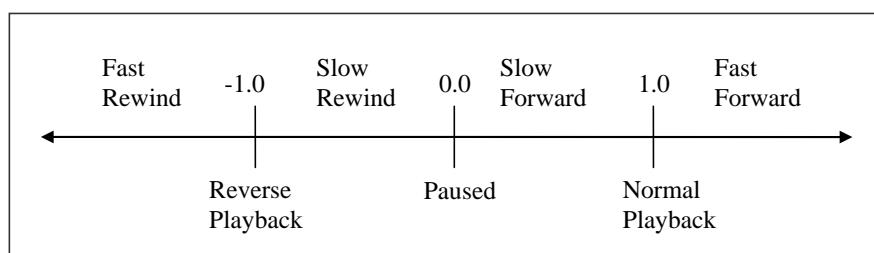


Figure 6-1. Mapping Time Scale Factors to Trick Modes

The IL client queries and sets the media clock's scale via the `OMX_IndexConfigTimeScale` configuration, passing the following structure:

```
typedef struct OMX_TIME_CONFIG_SCALETYP {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_S32 xScale;
} OMX_TIME_CONFIG_SCALETYP;
```

The clock component's client components are notified of changes in scale via their clock ports (see Clock Ports section for details).

### 6.2.2.2 Client Start Time

When a client is sent a start time (i.e., the timestamp of a buffer marked with the OMX\_BUFFERFLAG\_STARTTIME flag), it sends the start time to the clock component via OMX\_SetConfig on OMX\_IndexConfigTimeClientStartTime. This action communicates to the clock component the following information about the client's data stream:

- The stream is ready.
- The starting timestamp of the stream, either at startup or after a seek.

The clock component maintains a start time for every client component via a set of OMX\_TIME\_CONFIG\_TIMESTAMPTYPE structures. When transitioned to OMX\_TIME\_ClockStateWaitingForStartTime, the clock component waits on all start times prescribed by the transition. This ensures proper synchronization at the beginning of playback.

### 6.2.2.3 Media Clock State

The following structure represents the state of the clock component's media clock:

```
typedef struct OMX_TIME_CONFIG_CLOCKSTATETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_TIME_CLOCKSTATE eState;
    OMX_TICKS nStartTime;
    OMX_TICKS nOffset;
    OMX_U32 nWaitMask;
} OMX_TIME_CONFIG_CLOCKSTATETYPE;
```

The nStartTime field specifies the media time when the clock was started or will be started.

The nWaitMask field is a bit mask specifying the client components that the clock component will wait on in the OMX\_TIME\_ClockStateWaitingForStartTime state. Bit masks are defined as OMX\_CLOCKPORT0 through OMX\_CLOCKPORT7.

The nOffset field specifies the time by which to offset the media time. The clock component factors this value into the calculation of media time, effectively adding the offset to the media time reported to its clients. For example, a nOffset value of -x implies a pre-roll of duration x.

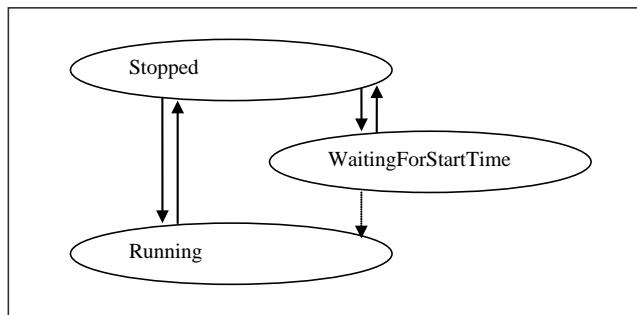
The eState field contains one of the possible clock state values shown in Table 6-2:

Table 6-2: Clock State Values

OMX_TIME_CLOCKSTATE Value	Interpretation
OMX_TIME_ClockStateRunning	The media clock is running.
OMX_TIME_ClockStateWaitingForStartTime	The media clock is waiting to run until all designated clients emit their start time.
OMX_TIME_ClockStateStopped	The media clock is stopped.

An OMX\_GetConfig execution using index OMX\_IndexConfigTimeClockState and structure OMX\_TIME\_CONFIG\_CLOCKSTATETYPE queries the current clock state.

An OMX\_SetConfig execution using index OMX\_IndexConfigTimeClockState and structure OMX\_TIME\_CONFIG\_CLOCKSTATETYPE commands the clock component to transition to the given state, effectively providing the IL client a mechanism for starting and stopping the media clock. Figure 6-2 shows the clock state transitions.



**Figure 6-2. Clock State Transitions**

Upon receiving OMX\_SetConfig from the IL client that requests a transition to the given state, the clock component will do the following:

- OMX\_TIME\_ClockStateStopped: Immediately stop the media clock, clear all pending media time requests, clear and all client start times, and transition to the stopped state. This transition is valid from all other states.
- OMX\_TIME\_ClockStateRunning: Immediately start the media clock using the given start time and offset, and transition to the running state. This transition is valid from all other states.
- OMX\_TIME\_ClockStateWaitingForStartTime: Transition immediately to the waiting state, wait for all clients specified in nWaitMask to report their start time, start the media clock using the minimum of all client start times and transition to OMX\_TIME\_ClockStateRunning. This transition is only valid from the OMX\_TIME\_ClockStateStopped state.

### 6.2.3 Wall Clock

The clock component maintains its own free running wall clock. It uses the wall clock to extrapolate media time values from the periodic updates from the reference clock. An IL client may query the current wall time via the OMX\_IndexConfigTimeCurrentWallTime configuration.

### 6.2.4 Reference Clocks

*Clock component is ignorant of the reference clock provider. It is the responsibility of the IL client to select and inform the reference clock provider of its reference provider role.*

#### **6.2.4.1 Reference clock provider**

IL client sets the reference clock provider via `OMX_SetConfig` using `OMX_IndexConfigTimeActiveRefClockUpdate` configuration and the following structure:

```
typedef struct OMX_TIME_CONFIG_ACTIVEREFCLOCKUPDATETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_BOOL bEnableRefClockUpdates;
    OMX_TICKS nRefTimeUpdateInterval;
} OMX_TIME_CONFIG_ACTIVEREFCLOCKUPDATETYPE;
```

- `bEnableRefClockUpdates` field indicates that a clock client is a reference clock provider or not. A value of `OMX_TRUE` means that the clock client is responsible for providing reference clock updates to clock. `OMX_FALSE` as value indicates the clock client to stop reporting the reference time updates to clock.
- `nRefTimeUpdateInterval` field indicates the report frequency of reference time updates to clock in microseconds. Zero value indicates the clock clients to use its default value. This field is valid only when `bEnableRefClockUpdates` is set to `OMX_TRUE`.

All Clock clients by default shall not provide any reference time updates unless instructed by the IL client. It is recommended for IL client to make sure no two clock clients report reference time updates at same time. When IL client intends to switch the reference clock provider, it is recommended to disable the current reference clock provider before enabling the new provider.

A Clock client that is not capable of being a reference clock provider shall return `OMX_ErrorUnsupportedSetting` when it is instructed. IL client can decide to dismantle the graph or continue with the graph where clock component will work on wall clock.

In general, any time audio is rendered or captured, the IL client should prefer the audio reference clock. Otherwise, the IL client should prefer the video reference.

#### **6.2.4.2 Reference Clock updates**

The clock component can accept reference updates from clock clients. Each reference clock tracks the media time at its associated component (i.e., the timestamp of the data currently being processed at that component) and provides periodic references to the clock component via `OMX_SetConfig` using `OMX_IndexConfigTimeCurrentReference` and passing the `OMX_TIME_CONFIG_TIMESTAMPTYPE` structure

When the clock component receives a reference, it updates its internally maintained media time with the reference. This action synchronizes the clock component with the client that is providing the reference clock.

The clock component can accept both a video and an audio reference clock, supplied respectively by a video component and an audio component. Each reference clock tracks the media time at its associated component (i.e., the timestamp of the data currently being processed at that component) and provides periodic references to the clock component via `OMX_SetConfig` using `OMX_IndexConfigTimeCurrentAudioReference` and `OMX_IndexConfigTimeCurrentVideoReference` and passing the following structure:

```
typedef struct OMX_TIME_CONFIG_TIMESTAMPTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_TICKS nTimestamp;
} OMX_TIME_CONFIG_TIMESTAMPTYPE;
```

When the clock component receives a reference, it updates its internally maintained media time with the reference. This action synchronizes the clock component with the component that is providing the reference clock.

The IL client controls which reference clock the clock component uses (if any) via the `OMX_IndexConfigTimeActiveRefClock` configuration and the following structure:

```
typedef struct OMX_TIME_CONFIG_ACTIVEREFCLKTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_TIME_REFCLKTYPE eClock;
} OMX_TIME_CONFIG_ACTIVEREFCLKTYPE;
```

Possible `eClock` values include those shown in Table 6-3:

**Table 6-3: Reference Clock Values**

OMX_TIME_REFCLKTYPE Value	Interpretation
<code>OMX_TIME_RefClockNone</code>	Not using a reference clock
<code>OMX_TIME_RefClockAudio</code>	Using audio reference clock
<code>OMX_TIME_RefClockVideo</code>	Using video reference clock

In general, any time audio is rendered or captured, the IL client should prefer the audio reference clock. Otherwise, the IL client should prefer the video reference.

#### 6.2.4.16.2.4.3 Media Time Updates

A clock component sends its clients media time updates, which can be either the fulfillment of a request, or a scale or state change notification, over its clock port. The structure used is the `OMX_TIME_MEDIATIMETYPE` structure. This can be signaled using the `OMX_SetConfig` call, with the index `OMX_IndexConfigTimeUpdate`, or written into the payload of a buffer.

The first method, using `OMX_SetConfig`, is mandatory when the clock port is connected to another component using a tunnel. In this case, when the tunnel is created using `OMX_SetupTunnel`, the clock ports shall advertise `nBufferCountActual` to be zero, since no buffers are required on this port. When these ports are required to

allocate buffers, then since the buffer count is zero these ports shall be automatically populated.

If the port on the clock component is not tunneled, so directly connected to the client, then the default non-zero value of nBufferCountActual shall be used, so that buffers are allocated on the port, and used to send OMX\_TIME\_MEDIATIMETYPE structures.

A clock component sends a client a media time update, as either the fulfillment of a request or a scale change notification, over its clock port via the following structure:

```
typedef struct OMX_TIME_MEDIATIMETYPE {  
    OMX_U32 nSize;  
    OMX_VERSIONTYPE nVersion;  
    OMX_U32 nClientPrivate;  
    OMX_TIME_UPDATETYPE eUpdateType;  
    OMX_TICKS nMediaTimestamp;  
    OMX_TICKS nOffset;  
    OMX_TICKS nWallTimeAtMediaTime;  
    OMX_S32 xScale;  
    OMX_TIME_CLOCKSTATE eState;  
} OMX_TIME_MEDIATIMETYPE;
```

- If the eUpdateType field indicates this is a request fulfillment message, the nClientPrivate field contains the value of pClientPrivate from the OMX\_TIME\_CONFIG\_MEDIATIMEREQUESTTYPE structure used to signal the request that this message is fulfilling. If the eUpdateType field indicates this is scale or state change notification, the nClientPrivate field will be zero.
- eUpdateType indicates the reason for the update and as one of the values shown in Table 6-3:

**Table 6-3: Update Types**

OMX_TIME_UPDATETYPE Value	Interpretation
OMX_TIME_UpdateRequestFulfillment	Fulfillment of a media time request.
OMX_TIME_UpdateScaleChanged	Notification of a scale change.
OMX_TIME_UpdateClockStateChanged	Notification of a clock state change.

- The nMediaTimestamp field specifies the target media timestamp (if this is a request fulfillment).
- The nOffset field specifies the distance in wall time between the current time and the target time (if this is a request fulfillment).
- The nWallTimeAtMediaTime field specifies the wall time corresponding to the target media timestamp (if this is a request fulfillment).
- The xScale field contains the scale of the media clock when the structure was completed.

- The eState field contains the clock state of the media clock when the structure was completed.

#### **6.2.4.26.2.4.4 Media Time Request**

A client requests the transmission of a particular timestamp via OMX\_SetConfig on its clock port using the OMX\_IndexConfigTimeMediaTimeRequest configuration. The following structure encapsulates a request:

```
typedef struct OMX_TIME_CONFIG_MEDIATIMEREQUESTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_PTR pClientPrivate;
    OMX_TICKS nMediaTimestamp;
    OMX_TICKS nOffset;
} OMX_TIME_CONFIG_MEDIATIMEREQUESTTYPE;
```

The client's request includes a timestamp, which is usually associated with some operation (e.g., the presentation of a frame) that the client shall execute at that time. Conceptually, the clock component fulfills the request when the media time matches the timestamp specified.

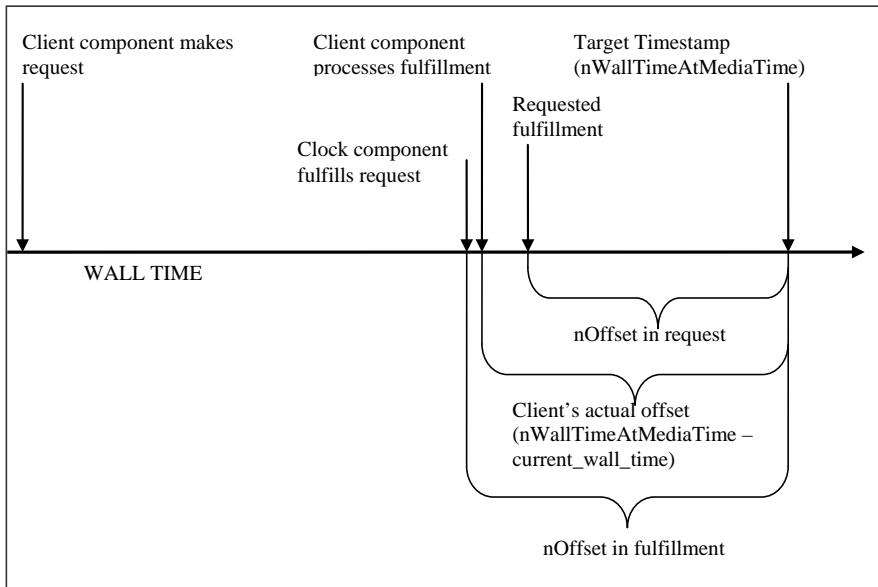
In practice, the client component may need the request fulfilled slightly earlier than the timestamp specified. In this case, the client specifies the earlier time need of the fulfillment via the nOffset field. nOffset specifies the desired difference between the wall time when the timestamp actually occurs and the wall time when the request is to be fulfilled. (The nOffset value should represent a relatively small interval, on the order of a few milliseconds.) Note that, due to the way scale modifies the progression of media time, a client cannot simply subtract the offset from the timestamp requested.

The request also includes a pointer to any private data that the client wants to associate with it (e.g., a pointer to the frame to deliver at the given timestamp).

#### **6.2.4.36.2.4.5 Media Time Request Fulfillment**

When fulfilling a request, the OMX\_TIME\_MEDIATIMETYPE structure contains the requested media time, the wall time that corresponds to that media time, and the offset in wall time between when the media time will actually occur and when the request was actually fulfilled.

Since some clock component implementations may have difficulty fulfilling the request at exactly the time specified, the fulfillment may occur slightly earlier, leading to a fulfillment offset larger than the one requested. The clock component shall fulfill the request as close to the requested time as possible without being late. Figure 6-3 shows the timeline for the request and fulfillment of a media time update.



**Figure 6-3. Timeline for Request and Fulfillment of Media Time Update**

When a client receives the fulfillment of a request, it may time any associated operation (e.g., frame delivery) more precisely by waiting any of the remaining interval until the timestamp itself. The client may estimate the interval until the timestamp actually occurs by using `nOffset` directly, although this does not account for any delay between when the clock component fulfilled the request and when the client began processing the fulfillment. A client may obtain a more accurate estimate for this interval by taking the difference between `nWallTimeAtMediaTime` and the clock component's current wall time, which is obtained via `OMX_GetConfig` on `OMX_IndexConfigTimeCurrentWallTime`.

This interval should be small enough for the client to use its own wall clock to implement the wait. The effect of any scale change during the interval or any drift between the clock component's wall clock and the client's wall clocks should be negligible for so short a duration.

#### **6.2.4.46.2.4.6 Scale Change Notifications**

A `eUpdateType` value of `OMX_TIME_UpdateScaleChanged` identifies a media time update as a scale change notification.

The clock component alerts its clients to scale changes via media time updates for optimization and data correction. For instance, during fast forward, a video component might skip intra frames and an audio component might scale and pitch correct its samples or drop them entirely. Nevertheless, components should never alter the presentation timestamp associated with a media sample. Time scaling is always applied to the media time, not the media samples.

A component that provides a reference clock shall watch for scale changes and behave accordingly. In particular, it shall:

- Cease all data delivery and its reference clock when the scale is zero (i.e., paused).
- Resume data delivery and its reference clock when the scale changes to non-zero (i.e., unpaused).

The xScale field contains the new scale. The nMediaTimestamp and nWallTimeAtMediaTime fields contain the media and wall time, respectively, when the scale change occurred. nOffset should reflect the difference, if any, between the wall time of the scale change and the wall time of the transmission of the corresponding media time update.

#### **6.2.4.56.2.4.7 Clock State Change Notifications**

A eUpdateType value of OMX\_TIME\_UpdateClockStateChanged identifies a media time update as a scale change notification.

The clock component alerts its clients to clock state transitions via media time updates so that they may take any action appropriate in that clock state. In particular:

- Any rendering component shall cease data delivery when the media clock transitions into the stopped state.
- Any client providing a reference clock shall use a media time request to time the resumption of data delivery and, hence, its reference clock when the media clock transitions into the running state

The eState field contains the new clock state. The nMediaTimestamp and nWallTimeAtMediaTime fields contain the media and wall time, respectively, when the clock change occurred. nOffset should reflect the difference, if any, between the wall time of the state change and the wall time of the transmission of the corresponding media time update.

#### **6.2.5 Rendering Delay**

Clock should have to accommodate for the rendering delay of clock clients before starting the media clock to provide a proper audio / video synchronization.

Additionally Clock shall maintain per client rendering delay to use as offset for all media time requests. Offsets in each media time requests shall be used in addition to the rendering delay of that clock client. Most use cases would not require offset per media time request.

Clock can query the rendering delay of the clients using OMX\_IndexConfigTimeRenderingDelay configuration and the following structure:

```
typedef struct OMX_TIME_CONFIG_RENDERINGDELAYTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
```

```
    OMX_U32 nPortIndex;
    OMX_TICKS nRenderingDelay;
} OMX_TIME_CONFIG_RENDERINGDELAYTYPE;
```

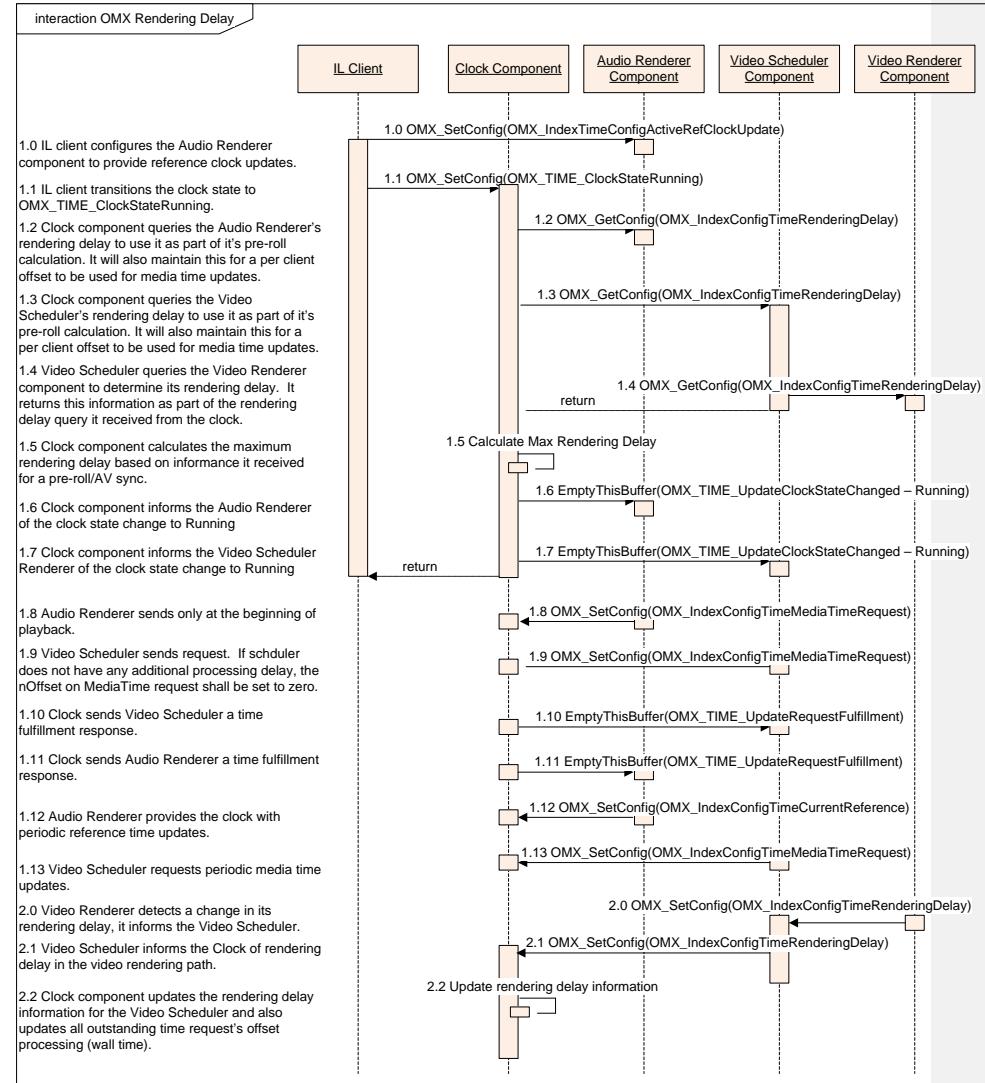
- nPortIndex represents the port that this structure applies to.
- nRenderingDelay field indicates rendering delay in microseconds for the clock client on the nPortIndex port.

Clock can default the value to zero for clients that do not support the OMX\_IndexConfigTimeRenderingDelay config. Clock component can query this config from clients either during the state transition to OMX\_TIME\_ClockStateRunning or OMX\_TIME\_ClockStateWaitingForStartTime.

Renderers can use this mechanism to propagate any change in rendering delay to the clock. In case of video domain, video renderer should inform the scheduler which in turn SHALL update the rendering delay to the clock. Clock on receipt of a new rendering delay shall update all outstanding requests for that client based on the new value. The rendering delay value shall be updated by OMX\_SetConfig with the index param OMX\_IndexConfigTimeRenderingDelay.

Other special clock client components like demuxers, etc which is not involved in rendering operation directly can either opt not to support the config or return to zero as rendering delay.

#### 6.2.5.1 Example



**Figure 6-4: OpenMAX Rendering Delay**

### 6.2.56.2.6 Clock Component Implementation

The clock component is responsible for implementing the semantics described in this section. Specifically the clock component should implement the following:

- Queries of its wall or media clock
- Queries of or changes to its media clock's state or scale

- Queries of or changes to its active reference clock
- Client notification of scale changes
- Fulfillment of media time requests
- Updates from the reference clocks

This following discussion describes aspects of these obligations that are not implicit in the preceding description of clock component semantics.

#### **6.2.5.16.2.6.1 Deriving Media Time**

The clock component derives the media time from the reference clock and the wall clock. When the reference clock sends the clock component a time reference,  $R_{now}$ , the clock component queries the wall clock for its current value,  $W_{now}$ .

The rendering delay has to be adjusted into the media time calculation. If an IL client specified an offset when it started the clock component (e.g., to implement a pre-roll), then the clock component calculates the ClockOffset in the following way

MaxRenderingDelay is the maximum of delay across all clients.

ClientOffset is the value provided by IL client

```
If( ClientOffset < 0 )
    ClockOffset = MIN(ClientOffset, -MaxRenderingDelay)
Else
    ClockOffset = ClientOffset – MaxRenderingDelay
```

With negative ClockOffset ( used for preroll & Rendering delay )

When clock starts, (Note that wall clock starts running, but not media clock)

$R_{base} \equiv 0$

$W_{base} = W_{now} + ClockOffset$

Reference time updates

The clock component stores the ultimate reference/wall time pair, representing the base of extrapolation, for later use as  $\langle R_{base}, W_{base} \rangle$  where:

$R_b = R_{now}$

$W_b = W_{now}$

Media time is calculated using the following:

```
If( MediaClock started )
    Mnow = Rbase + scale * (Wnow – Wbase)
Else
```

#### // Initialized Mnow value

The clock component derives the media time from the reference clock and the wall clock. When the reference clock sends the clock component a time reference,  $R_{now}$ , the clock component queries the wall clock for its current value,  $W_{now}$ . If an IL client specified an offset when it started the clock component (e.g., to implement a pre-roll), then the clock component adds this offset as  $W_{now} + \text{Offset}$ . The clock component stores the ultimate reference/wall time pair, representing the base of extrapolation, for later use as  $\langle R_{base}, W_{base} \rangle$  where:

$$R_{base} = R_{now}$$

$$W_{base} = W_{now} + \text{Offset}$$

The clock component calculates the instantaneous media time,  $M_{now}$ , by querying the wall clock,  $W_{now}$ , and extrapolating from the last reference, modulated by the current scale,  $Scale$ , as follows:

$$M_{now} = R_{base} + Scale * (W_{now} - W_{base})$$

#### 6.2.5.26.2.6.2 Scale Changes

Upon invocation of a scale factor,  $Scale$ , the clock component first establishes a new base of extrapolation by querying the current media time,  $M_{now}$ , and the current wall time,  $W_{now}$ :

$$R_{base} = M_{now}$$

$$W_{base} = W_{now}$$

The clock component then notifies all client components of the new scale via a media time update. It fills in the fields of the corresponding OMX\_TIME\_MEDIATIMETYPE structure as follows:

- nClientPrivate = NULL
- nMediaTimestamp =  $M_{now}$
- nWallTimeAtMediaTime =  $W_{now}$
- xScale =  $Scale$

#### 6.2.5.36.2.6.3 Fulfilling Media Time Requests

A clock component's approach to servicing media time requests is implementation specific. Certain operating system constructs (e.g., timers) may be useful in avoiding the expense of the spin locks associated with comparing requested times with the current media time. Nevertheless, clock component implementers should be wary of any skew between the clock component and the clock used by the operating system constructs that compromise the timely, accurate fulfillment of requests.

The clock component shall account for any offset specified by the request. Assume a requested timestamp of  $M_{request}$ , an offset  $Offset_{request}$ , and a scale factor of  $Scale$ . Instead of comparing against  $M_{request}$ , the clock component should compare against the following:

$$M_{request} - (Offset_{request} * Scale)$$

Furthermore, the comparison between requested times and media time differ between forward playback, backward, and paused playback. Specifically, the comparisons shown in Table 6-4 should be used according to scale:

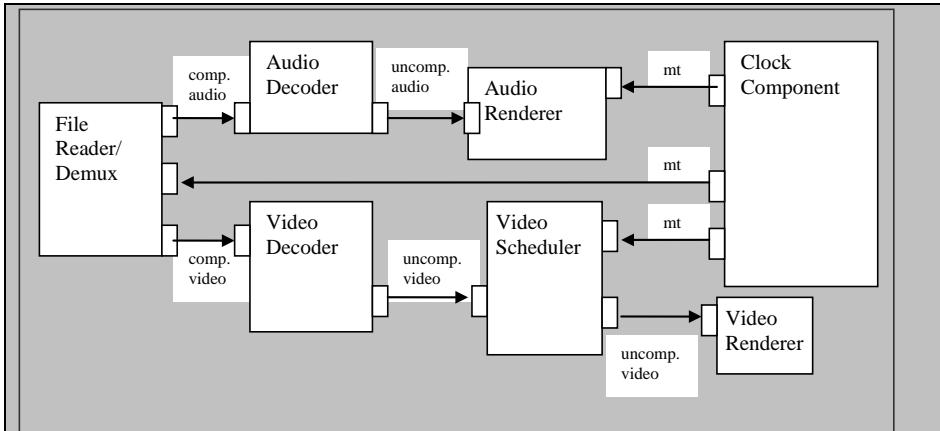
**Table 6-4: Media Time Request Scale**

Scale	Fulfill request when
> 0.0 (forward playback)	$M_{now} \geq (M_{request} - (Offset_{request} * Scale))$
< 0.0 (backward playback)	$M_{now} \leq (M_{request} - (Offset_{request} * Scale))$
0.0 (paused)	Never

#### **6.2.66.2.7 Audio-Video File Playback Example Use Case**

As an example, examine the playback of a file containing synchronized audio and video as illustrated in [Figure 6-5](#). This example assumes that each audio or video frame has a presentation timestamp associated with it. In this construction, a file reader/demultiplexing component feeds compressed audio and video streams to a pair of decoders. The decoders send uncompressed data to an audio renderer and video scheduler. The audio renderer delivers data to the hardware and the video scheduler will send the data to the video renderer which will send the data to the hardware.

The audio renderer and video scheduler coordinate with the clock component to implement smooth synchronized audio-video delivery. The audio renderer, video scheduler and file demuxer are clients of the clock component (connected on their respective clock ports) so they may watch for scale changes. The video scheduler also uses the clock component to time delivery of video frames via media time requests.



**Figure 6-5. Example Use Case of Audio-Video File Playback**

The audio renderer and video scheduler act as the audio and video reference clocks, each sending their reference times to the clock component as they deliver data.

In this example, the IL client uses the audio renderer as the reference clock at any time audio data is being delivered during normal playback. Thus, the IL client does not need to use the clock component to coordinate the delivery of audio data. It simply feeds new data to the audio device whenever it can, provided that the current scale allows it. When the audio device is presenting an audio buffer, the audio renderer emits the timestamp of that buffer as a reference.

The video scheduler, however, shall coordinate with the clock component when delivering video frames. For each frame that the video scheduler will deliver the frame to the video renderer at a particular timestamp, the following occurs:

1. The video scheduler submits a media time request, referencing the frame data in the private pointer and specifying fulfillment slightly earlier than the timestamp.
2. The clock component fulfills the request when it becomes current via a media time update to the video scheduler that references the original timestamp and includes the private pointer.
3. The video scheduler receives the media time update, de-references the private pointer to obtain the frame data, and delivers the frame to the video renderer. The video scheduler uses an implementation-specific mechanism to wait the remainder of the time until the timestamp before delivery (e.g., schedules a hardware flip with the video driver).

The IL client controls the clock component via specialized configurations to start and stop the media clock. To implement trick modes, the IL client sets the scale factor configuration. When the clock component applies the scale to the calculation of media time, it sends a media time update with the scale change to all of its clients.

The client components react to that scale change appropriately. When the scale is 0 (i.e., the media clock is paused), the audio renderer silences audio and ceases sending data.

Furthermore, in this example, the file demuxer might elect to ignore input during non-1X playback.

If audio is effectively silenced during trick modes, the IL client may switch the active reference clock from the audio reference to the video reference.

Finally, the IL client may query the current media time from the clock component to, for instance, update the user interface such as through a progress bar.

## 7 Container Parsing

This section describes container parsing including access to available streams and metadata.

### 7.1 Parameter and Configuration Indexes

The header OMX\_Index.h contains the enumeration OMX\_INDEXTYPE, which contains all of the standard index values used with the functions OMX\_GetParameter, OMX\_SetParameter, OMX\_GetConfig, and OMX\_SetConfig. Table 7-1 describes the index values that relate to file parsing.

**Table 7-1: Index Values for File Parsing**

Index	Description
OMX_IndexParamNumAvailableStreams	Specifies the number of alternative streams available on a given output port.  The corresponding structure is OMX_PARAM_U32TYPE.
OMX_IndexParamActiveStream	Specifies the active stream (among those available) on a given output port.  The corresponding structure is OMX_PARAM_U32TYPE.
OMX_IndexParamMetadataKeyFilter	Specifies whether a key (or all keys) are enabled or disabled with respect to the metadata filter. An enabled key is in the filter and metadata with this key is retained for future potential querying.  The corresponding structure is OMX_PARAM_METADATAFILTERTYPE.
OMX_IndexConfigMetadataItemCount	Specifies number of metadata items associated with a resource contained within a media file at a specific scope.  The corresponding structure is OMX_CONFIG_METADATAITEMCOUNTTYPE.
OMX_IndexConfigMetadataItem	Specifies the contents of the metadata item indicated by the given index or key.  The corresponding structure is OMX_CONFIG_METADATAITEMTYPE.

Index	Description
OMX_IndexConfigContainerNodeCount	<p>Specifies the number of child nodes a given node contains.</p> <p>The corresponding structure is OMX_CONFIG_CONTAINERNODECOUNTTYPE.</p>
OMX_IndexConfigCounterNodeID	<p>Specifies the node id of specific node.</p> <p>The corresponding structure is OMX_CONFIG_CONTAINERNODEIDTYPE.</p>
OMX_IndexParamMetadataFilterType	Specifies the filters to be applied for the meta data accesses

## 7.2 Format Detection

A particular container parser implementation supports a finite set of container formats, yet the component might not definitively determine support for a particular datastream until it attempts to parse the datastream. Therefore OpenMAX IL introduces the following mechanisms for a parser to communicate its ability or inability to recognize the format of a given datastream:

- The OMX\_ErrorFormatNotDetected error. A component sends the client this error (in the form of an OMX\_EventError event passed via the EventHandler callback) when it cannot parse or determine the format of the given datastream.
- The OMX\_EventPortFormatDetected event. A component sends the client this event (via the EventHandler callback) when it has successfully recognized a format and determined that it can support it.

The IL client may use these mechanisms (perhaps in conjunction with autodetect ports) to determine whether a given parser is appropriate for a given datastream.

## 7.3 Port Streams

When parsing a datastream a component may discover multiple alternative streams suitable for emission as output on a given output port. For instance, when parsing a video stream muxed with synchronized audio, a parser component may discover the container datastream includes several alternative languages represented as different audio streams each a candidate for output out the same audio output port.

A port exposes the set of candidate streams as a “port stream”. If a port supports port streams (e.g. a parser output port), discovering the port streams is part of that port’s autodetect process. When the autodetect is completed (i.e. the component issues a OMX\_EventPortSettingsChanged event) such a port **beis** ready to service queries and writes on the following configs:

- The OMX\_IndexParamNumAvailableStreams config. This read only parameter denotes the number of streams available on the port.
- The OMX\_IndexParamActiveStream config. This read/write parameter denotes the currently selected stream for the port.

The port populates its settings according to the currently selected stream. An IL client may use thus use the OMX\_IndexParamActiveStream parameter to both browse the settings associated with each available streams and to ultimately select the final stream for playback.

This may be performed by the IL client in the following way:

1. Instantiate the component and set any relevant configs/parameters (e.g. identifying the target content)
2. Set all output ports where the IL client desires stream discovery to autodetect and put the component into the OMX\_StateExecuting state.
3. Wait until the port generates an OMX\_EventPortSettingsChanged event. This event indicates it has parser enough data to have discovered the alternative streams.
4. Query the number of available streams for that port via OMX\_IndexParamNumAvailableStreams. For each possible stream set that stream as active via OMX\_IndexParamActiveStream. This will cause the port to populate its settings according to the active stream. The IL client may then discover the properties of the stream by reading the appropriate port parameters.
5. After reading the properties of each stream, the IL client may select the one it desires via OMX\_IndexParamActiveStream.

## 7.4 Metadata Extraction

OpenMAX IL supports retrieving metadata items captured by a component. A metadata item is defined as a key/value pair, where both key and value are buffers formatted using specified character sets. OpenMAX IL enables an [IL client](#) to perform the following operations with regards to metadata:

- Specify an client-defined set of keys to filters which metadata items will be captured by the component
- Scope a metadata query to seek particular elements of the content, inclusive of the entire content
- Determine the number of distinct metadata items available at any given scope
- Retrieve all metadata items by iterating through all metadata items by available at any given scope by index
- Retrieve a metadata value for a specific metadata key

#### **7.4.1.1 Key/Value Query**

OpenMAX IL supports the querying of key/value pair data captured by a component that parses metadata via a set of component configs. The purpose of these configs is to enable an [IL client](#) to determine how many metadata items are present at a given scope, iterate through the items by index to retrieve the key/value data and query values for specific keys.

#### **7.4.1.2 Node Traversal**

OpenMAX IL supports the traversal of metadata nodes captured by a component that parses metadata via a set of component configs.

The purpose of these configs is to define a mechanism for obtaining a set of specifiers which can be used to uniquely scope metadata searches to atomic elements, or ‘nodes’, of data within a media file. Each node has a component-defined ‘node ID’ that the component can use to uniquely locate the node within the media file. Note that a node ID should be considered an opaque ID, therefore it need not have any intrinsic value or meaning; it need only be a value that the component can use to uniquely set the scope of a metadata search.

All media files contain exactly one ‘root node’ whose node ID always has value OMX\_ALL; this represents the ‘top-level’ metadata associated with the media file. The root node is the only node without a parent node. All other nodes have exactly one parent.

In general, the node traversal configs uses the term ‘node’ is used to represent a node for which one wants to know the ID value, and the term ‘parent node’ is used to represent the parent of one or more nodes for which one wants to know the ID value(s).

#### **7.4.1.3 Key Filtering**

OpenMAX IL supports the filtering of metadata captured by a component that parses metadata via the OMX\_IndexParamMetadataKeyFilter parameter. This parameter allows the client to add or remove keys from the filter before the component begins processing the data. A component will retain all metadata associated with keys in the filter (so the IL client may query them later) and may safely ignore all keys not in the filter.

#### **7.4.1.4 Specifying Language/Country**

The concepts of Language and Country for a metadata item exist in some but not all file format metadata schemes. Where they do exist, most formats have only Language (including ID3v2), whereas others combine Language and Country together into a single, compound specifier. Only 3GPP has a standard metadata key that uses a Country specifier but no Language (in ‘loc’ metadata items).

Because of the relatively rare usage of these features, at the API level we combine Language and Country into a single compound Language-Country specifier, where Language comes first and Country is optional, as per the HTTP specification (RFC 2068). This approach accommodates all use cases; for example, “en” indicates English language

content for all countries, “en-US” indicates English language content for the US, “en-UK” indicates English language content for the UK, etc.

Individual requirements for Language and Country follow.

#### **7.4.1.4.1 Language Codes**

When accessing the value of a metadata item for which a language is specified, the client shall be given the language specifier. When creating a metadata item for which a language may be specified, or when changing its value, the client shall be able to indicate the language used in the supplied value. This is necessary because some file formats allow some metadata items to include a language specifier (this is usually limited to text, though not necessarily; for example, images and sounds can also be in a particular language). In some cases, there may be multiple, alternative versions of the same metadata item in different languages, and in these cases the language specifier allows the client application to select and present just the most appropriate version.

Public standards for Language specifiers include RFC 1766 / ISO 639.

#### **7.4.1.4.2 Country Codes**

Similar to the Language requirement: When accessing the value of a metadata item for which a Country (geographic location) is specified, the client shall be given the Country specifier. When creating a metadata item for which a Country may be specified, or when changing its value, the client shall be able to indicate the Country to which the supplied value applies.

Public standards for Country specifiers include ISO 3166.

### **7.5 Types and Structures**

#### **7.5.1 OMX\_PARAM\_U32TYPE**

Parameters represented by unsigned 32 bit values (e.g. OMX\_IndexParamActiveStream) use the OMX\_PARAM\_U32TYPE which is defined as follows:

```
typedef struct OMX_PARAM_U32TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nU32;
} OMX_PARAM_U32TYPE;
```

#### **7.5.2 OMX\_METADATACHARSETTYPE**

The OMX\_METADATACHARSETTYPE enumeration defines the range of possible character sets (e.g. where a particular character is used to represent a metadata key).

**Table 7-2: Supported Metadata Charset Types**

<b>Value Name</b>	<b>Character Set Description</b>
OMX_MetadataCharsetUnknown	Unknown character encoding
OMX_MetadataCharsetASCII	ASCII
OMX_MetadataCharsetBinary	Binary
OMX_MetadataCharsetCodePage1252	Microsoft Code Page 1252
OMX_MetadataCharsetUTF8	Unicode UTF-8
OMX_MetadataCharsetJavaConformantUTF8	Unicode UTF-8 (Java Conformant)
OMX_MetadataCharsetUTF7	Unicode UTF7
OMX_MetadataCharsetImapUTF7	Unicode UTF-7 per IETF RFC 2060
OMX_MetadataCharsetUTF16LE	Unicode UTF-16 (Little Endian)
OMX_MetadataCharsetUTF16BE	Unicode UTF-16 (Big Endian)
OMX_MetadataCharsetGB12345	GB 12345 (Chinese)
OMX_MetadataCharsetHZGB2312	HZ GB 2312 (Chinese)
OMX_MetadataCharsetGB2312	GB 2312 (Chinese)
OMX_MetadataCharsetGB18030	GB 18030 (Chinese)
OMX_MetadataCharsetGBK	GBK (CP936) (Chinese)
OMX_MetadataCharsetBig5	Big 5 (Chinese)
OMX_MetadataCharsetISO88591	ISO-8859-1 (Latin1 – West European languages)
OMX_MetadataCharsetISO88592	ISO-8859-2 (Latin2 – East European)
OMX_MetadataCharsetISO88593	ISO-8859-3 (Latin3 – South European)
OMX_MetadataCharsetISO88594	ISO-8859-4 (Latin4 – North European)
OMX_MetadataCharsetISO88595	ISO-8859-5 (Cyrillic)
OMX_MetadataCharsetISO88596	ISO-8859-6 (Arabic)
OMX_MetadataCharsetISO88597	ISO-8859-7 (Greek)
OMX_MetadataCharsetISO88598	ISO-8859-8 (Hebrew)
OMX_MetadataCharsetISO88599	ISO-8859-9 (Latin5 - Turkish)
OMX_MetadataCharsetISO885910	ISO-8859-10 (Latin6 – Nordic)
OMX_MetadataCharsetISO885913	ISO-8859-13 (Latin7 – Baltic Rim)
OMX_MetadataCharsetISO885914	ISO-8859-14 (Latin8 - Celtic)
OMX_MetadataCharsetISO885915	ISO-8859-15 (Latin9 – updates to Latin1)
OMX_MetadataCharsetShiftJIS	Shift-JIS (Japanese)
OMX_MetadataCharsetISO2022JP	ISO-2022-JP (Japanese)
OMX_MetadataCharsetISO2022JP1	ISO-2022-JP-1 (Japanese)
OMX_MetadataCharsetISOEUCJP	ISO EUC-JP (Japanese)
OMX_MetadataCharsetSMS7Bit	SMS 7-bit

### 7.5.3 OMX\_METADATASCOPETYPE

The OMX\_METADATASCOPETYPE structure is used to identify the type of the metadata search scope that is being specified. A scope type value is used in conjunction with a scope specifier value to identify the type of said specifier.

**Table 7-3: Supported Metadata ScopeTypes**

Value Name	Client usage	Component action
OMX_MetadataScopeAllLevels	Search entire piece of content—scope specifier is ignored	Search entire piece of content for matching metadata.
OMX_MetadataScopeTopLevel	Limit search scope to root level—scope specifier is ignored	Search only at the content's root level for matching metadata. Root level is defined as the only container level with no logical parent.
OMX_MetadataScopePortLevel	Limit search scope to port level—scope specifier is the port index for an output port	Search for matches only among those metadata items associated with the media resource being emitted from the indicated port. If multiple streams can be emitted from the indicated port, the component will only search for matching metadata associated with the currently active stream, as determined using the port streams mechanism.
OMX_MetadataScopeNodeLevel	Limit search scope to container file node level—scope specifier is a node ID.	Search for matches only among those metadata items explicitly associated with the specified container node and exclusive of sub-nodes of the specified container node.

### 7.5.4 OMX\_CONFIG\_METADATAITEMCOUNTTYPE

The [IL client](#) uses the OMX\_IndexConfigMetadataItemCount and the OMX\_CONFIG\_METADATAITEMCOUNTTYPE structure to query a component for the number of metadata items associated with a resource contained within a media file at a specific scope.

OMX\_CONFIG\_METADATAITEMCOUNTTYPE is defined as follows.

```
typedef struct OMX_CONFIG_METADATAITEMCOUNTTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_METADATASCOPETYPE eScopeMode;
    OMX_U32 nScopeSpecifier;
```

```

    OMX_U32 nMetadataItemCount;
} OMX_CONFIG_METADATAITEMCOUNTTYPE;

```

#### 7.5.4.1 Parameter Definitions

The parameters for OMX\_CONFIG\_METADATAITEMCOUNTTYPE are defined as follows.

- eScopeMode defines the type of scope being specified. See Section [10—Implementing Buffer Sharing](#) for usage.
- nScopeSpecifier is the value of the scope specifier. See Section [10—Implementing Buffer Sharing](#) for usage.
- nMetadataItemCount is the number of metadata items found at the scope being queried.

#### 7.5.4.2 Dependencies

The OMX\_CONFIG\_METADATAITEMCOUNTTYPE structure may be queried at any time as generally allowed when calling `OMX_GetConfig`. However, it is possible the count of metadata items at a given scope may change as the data being processed by the component changes.

#### 7.5.4.3 Functionality

The OMX\_CONFIG\_METADATAITEMCOUNTTYPE structure identifies the number of metadata items in a particular scope.

#### 7.5.4.4 OMX\_METADATASEARCHMODETYPE

The OMX\_METADATASEARCHMODETYPE enumeration lists the types of queries that can be performed using the OMX\_CONFIG\_METADATAITEMTYPE structure.

As such the search mode specifies the usage of the other fields (input and output) of this configuration structure.

**Table 7-4: Supported Metadata Search Types**

Value Name	Client usage	Component action
OMX_MetadataSearchValueSizeByIndex	Get metadata value size by index nMetadataItemIndex = valid index for the given scope	nValueMaxSize = number of bytes needed to hold value of the found metadata item (No actual Key or Value data are returned, only the size.)

Value Name	Client usage	Component action
OMX_MetadataSearchItem ByIndex	<p>Get metadata key and value by index</p> <p>nMetadataItemIndex = valid index for the given scope</p> <p>nValueMaxSize = size in bytes of nValue buffer.</p> <p>nValue = empty buffer at least nValueMaxSize bytes long</p> <p>(Key buffer has fixed size.)</p>	<p>eKeyCharset = charset of key data in nKey</p> <p>nKeySizeUsed = number of bytes used in nKey</p> <p>nKey = buffer containing key data from the found metadata item</p> <p>eValueCharset = charset of value data in nValue</p> <p>nValueSizeUsed = number of bytes used in nValue</p> <p>nValue = buffer containing value data from the found metadata item</p>
OMX_MetadataSearchNextItem ByKey	<p>Get value of first, nth, or next metadata item matching a given key</p> <p>nMetadataItemIndex = Valid index for the given scope. To obtain the Nth occurrence of the key, set to N - 1. To obtain the first occurrence of the key, set to OMX_ALL.</p> <p>eKeyCharset = charset of key data in nData</p> <p>nKeySizeUsed = number of bytes used in nKey</p> <p>nKey = buffer containing the key data to match</p> <p>nValueMaxSize = size in bytes of allocated by client to receive value data</p> <p>nValue = empty buffer at least nValueSize bytes long</p>	<p>nMetadataItemIndex = index of matching/found metadata item</p> <p>eValueCharset = charset of value data in nValue</p> <p>nValueSizeUsed = number of bytes used in nValue</p> <p>nValue = buffer containing value data from the found metadata item</p>

### 7.5.5 OMX\_CONFIG\_METADATAITEMTYPE

The [IL\\_client](#) uses the OMX\_IndexConfigMetadataItem and the OMX\_CONFIG\_METADATAITEMTYPE structure to query a component for one metadata item. It can be used to retrieve a metadata item either by index or by key, or to get the size of a metadata item by index.

```

typedef struct OMX_CONFIG_METADATAITEMTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_METADATASCOPETYPE eScopeMode;
    OMX_U32 nScopeSpecifier;
    OMX_U32 nMetadataItemIndex;
    OMX_METADATASEARCHMODETYPE eSearchMode;
    OMX_METADATACHARSETTYPE eKeyCharset;
    OMX_U8 nKeySizeUsed;
    OMX_U8 nKey[128];
    OMX_METADATACHARSETTYPE eValueCharset;
    OMX_U8 sLanguageCountry[128];
    OMX_U32 nValueMaxSize;
    OMX_U32 nValueSizeUsed;
    OMX_U8 nValue[1];
} OMX_CONFIG_METADATAITEMTYPE;

```

### 7.5.5.1 Parameter Definitions

The parameters for OMX\_CONFIG\_METADATAITEMTYPE are defined as follows.

- eScopeMode defines the type of scope being specified.
- nScopeSpecifier is the value of the scope specifier.
- nMetadataItemIndex is the index of the metadata item being queried.
- eSearchMode is the type of query being performed.
- eKeyCharset is the OMX\_METADATACHARSETTYPE of the key data within nKey.
- nKeySizeUsed is number of bytes within nKey that are populated with key data.
- nKey is the buffer of key data.
- eValueCharset is the OMX\_METADATACHARSETTYPE of the value data within nValue.
- sLanguageCountry is the combined language and country specifier.
- nValueMaxSize is the size in bytes of the nValue buffer. **Note:** when nValueMaxSize is an input parameter and is a value less than the size of the metadata value, an OMX\_ErrorInsufficientResources error will be returned and no output parameters will be populated.
- nValueSizeUsed is the number of bytes within nValue that are populated with value data.
- nValue is the buffer of value data.

### 7.5.5.2 Dependencies

The OMX\_CONFIG\_METADATAITEMTYPE structure may be queried at any time as generally allowed when calling OMX\_GetConfig. However, it can be possible that the metadata item being sought may not yet be accessible if the corresponding portion of content has not yet been processed by the component.

### 7.5.5.3 Functionality

The OMX\_CONFIG\_METADATAITEMTYPE structure identifies a particular metadata item in a particular scope. The type of query performed by OMX\_GetParameter is defined by the eSearchMode field. Refer to Section 7.5.4.4 above for details.

## 7.5.6 OMX\_PARAM\_METADATAFILTERTYPE

The [IL client](#) uses the OMX\_IndexParamMetadataFilterType and OMX\_PARAM\_METADATAFILTERTYPE parameter structure to specify the inclusion or exclusion of a particular key, or of all keys using a given character set, in a component's filter of metadata keys. An IL client leverages writes to this parameter to enable or disable a particular key or key character set, which effectively includes or excludes that key or key character set from the set of metadata retained by the component for querying later. An IL client may also leverage reads of this parameter to query the for the inclusion/exclusion of keys from this filter. Metadata items may also be optionally filtered for Language/Country code in combination with a particular key or key character set.

```
typedef struct OMX_PARAM_METADATAFILTERTYPE
{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_BOOL bAllKeys;
    OMX_METADATACHARSETTYPE eKeyCharset;
    OMX_U32 nKeySizeUsed;
    OMX_U8 nKey [128];
    OMX_U32 nLanguageCountrySizeUsed;
    OMX_U8 nLanguageCountry[ 128 ];
    OMX_BOOL bEnabled;
} OMX_PARAM_METADATAFILTERTYPE;
```

### 7.5.6.1 Parameter Definitions

The parameters for OMX\_PARAM\_METADATAFILTERTYPE are defined as follows.

- nVersion is the version of the structure.
- nSize is the size of the structure in bytes. This value shall be specified when this structure is used as either an input to or output from a function.
- bAllKeys

If this field is false, then only the particular specified key is included in the filter, and the filter matches metadata items with the indicated language/country code (if present). None of the other fields are ignored.

If this field is true and nKeySizeUsed is zero and eKeyCharset is MetadataCharsetUnknown, then this structure refers to all possible keys in all possible eKeyCharsets, and matches metadata items with the indicated language/country codes (if present). The nKey field is ignored.

If this field is true and nKeySizeUsed is zero and eKeyCharset is not MetadataCharsetUnknown, then this structure refers to all possible keys in the specified eKeyCharset, and matches metadata items with the indicated language/country code (if present). The nKey field is ignored.

- eKeyCharset – If nKeySizeUsed is not zero, then this must be used to indicate the OMX\_METADATACHARSETTYPE of the key data within nKey. If nKeySizeUsed is zero, then all keys with this character set will be added to the filter; the value MetadataCharsetUnknown will match all key character sets.
- nKeySizeUsed is number of bytes within nKey that are populated with key data. If zero, there is no key associated with this metadata filter item (just an eKeyCharset and/or language/country code). If this is not zero, then the eKeyCharset must indicate the encoding of the key data in nKey.
- nKey is the buffer of key data.
- nLanguageCountrySizeUsed is the number of bytes within nLanguageCountry that are populated with Language / Country code data. If zero, there is no Language/Country code associated with this metadata filter item (just a key).
- nLanguageCountry is the buffer of Language/Country code data.
- bEnabled if true then key is part of filter (e.g. retained for query later). If false then key is not part of filter is the buffer of key data.

#### 7.5.6.2 Dependencies

The OMX\_PARAM\_METADATAFILTERTYPE structure may be queried at any time ~~that the component is not in the OMX\_StateInvalid state~~. The structure may be set using OMX\_SetParameter only when the component is in the OMX\_StateLoaded state.

#### 7.5.6.3 Functionality

The OMX\_PARAM\_METADATAFILTERTYPE structure identifies whether a particular metadata key or language/country code (or all metadata keys) are in the metadata filter (that is, they are retained by the parser for potential querying later). An IL client may thus leverage this structure and the OMX\_IndexParamMetadataKeyFilter parameter to set or get filter settings.

**Table 7-5: Meta Data Key Access Use Cases**

Use case	Function	bAllKeys	eKeyCharset nKeySizeUsed nKey, nLanguageCountry SizeUsed, nLanguageCountry	bEnabled
Add a key and/or language/country code to the filter	SetParameter	OMX_FALSE	Specifies particular key (and its encoding) being added to filter, with optional language/country code	OMX_TRUE
Add all keys to the filter (also matches language/country code, if any); if eKeyCharset is a known encoding, then only keys with that encoding are included in the filter	SetParameter	OMX_TRUE	Required: eKeyCharsetOptional: nLanguageCountrySizeUsed, nLanguageCountry. Others are not applicable/ignored	OMX_TRUE
Remove a key and/or language/country code from the filter	SetParameter	OMX_FALSE	Specifies particular key (and its encoding) being removed from filter, with optional language/country code	OMX_FALSE
Remove all keys from the filter (also matches language/country code, if any); if eKeyCharset is a known encoding, only keys with that encoding are included in the filter	SetParameter	OMX_TRUE	Required: eKeyCharset, Optional: nLanguageCountrySizeUsed, nLanguageCountry. Others are not applicable/ignored	OMX_FALSE
Query whether a key and/or language/country code is part of the filter	GetParameter	Not applicable/ignored	Specifies particular key (and its encoding) being queried, with optional language/country code	Output field filled in by GetParameter

#### 7.5.6.4 Post-processing Conditions

The changes specified to the component's metadata filter (i.e. the enabling or disabling of keys) are applied upon the return of a `OMX_SetParameter` call when used with the `OMX_CONFIG_METADATAITEMTYPE` structure. The component retains only the cumulative set of keys specified as enabled in the filter.

#### 7.5.7 `OMX_CONFIG_CONTAINERNODECOUNTTYPE`

The [IL client](#) uses the `OMX_IndexConfigContainerNodeCount` and the `OMX_CONFIG_CONTAINERNODECOUNTTYPE` structure to query a parent node for the number of nodes it contains.

```
typedef struct OMX_CONFIG_CONTAINERNODECOUNTTYPE
{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_BOOL bAllKeys;
    OMX_U32 nParentNodeID;
    OMX_U32 nNumNodes;
} OMX_CONFIG_CONTAINERNODECOUNTTYPE;
```

##### 7.5.7.1 Parameter Definitions

The parameters for `OMX_CONFIG_CONTAINERNODECOUNTTYPE` are defined as follows.

- `nParentNodeID` is the node ID for the node being queried. To specify the media file's root node, use the value `OMX_ALL`.
- `nNumNodes` is the number of nodes contained by the indicated parent node.

##### 7.5.7.2 Dependencies

The `OMX_CONFIG_CONTAINERNODECOUNTTYPE` structure may be queried at any time as generally allowed when calling `OMX_GetConfig`. However, it is possible that the count of nodes returned by this query may change if the component is actively processing data.

##### 7.5.7.3 Functionality

The `OMX_CONFIG_CONTAINERNODECOUNTTYPE` structure identifies the node count on given a node ID.

#### 7.5.8 `OMX_CONFIG_CONTAINERNODEIDTYPE`

The [IL client](#) uses the `OMX_IndexConfigCounterNodeID` and the `OMX_CONFIG_CONTAINERNODEIDTYPE` structure to obtain information about a specific node.

```

typedef struct OMX_CONFIG_CONTAINERNODEIDTYPE
{
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_BOOL bAllKeys;
    OMX_U32 nParentNodeID;
    OMX_U32 nNodeIndex;
    OMX_U32 nNodeID;
    OMX_U8STRING cnodeName[128];
    OMX_BOOL bIsLeafType;
} OMX_CONFIG_CONTAINERNODEIDTYPE;

```

#### 7.5.8.1 Parameter Definitions

The parameters for OMX\_CONFIG\_CONTAINERNODEIDTYPE are defined as follows.

- nParentNodeID is the node ID for the node being queried. To specify the media file's root node, use the value OMX\_ALL
- nNodeIndex is the index of this node.
- nNodeID is the node ID for this node.
- cnodeName name of this node. ~~It is an OMX\_STRING less than 128 characters long including the trailing null byte.~~
- bIsLeafType indicates whether this node may be a parent to other nodes. If the component does not know whether this node is a parent or not, the component will return OMX\_FALSE.

#### 7.5.8.2 Dependencies

The OMX\_CONFIG\_CONTAINERNODEIDTYPE structure may be queried at any time as generally allowed when calling OMX\_GetConfig. However, it is possible that if the underlying data has changed the node being sought may no longer be accessible.

#### 7.5.8.3 Functionality

The OMX\_CONFIG\_CONTAINERNODEIDTYPE structure identifies the properties of the node which is the specified child of the specified parent node.

## **8 Mandatory Component Parameters**

### **8.1 Component Role**

A component implementation may support one or more roles. We define a role as the behavior of component acting according to a particular standard or vendor-specific component definition. The name of the component definition identifies the role.

For example a given component implementation named “OMX.CompanyXYZ.MyAudioDecoder” might support the following roles:

audio\_decoder.mp3

audio\_decoder.aac

audio\_decoder.amr

When the audio\_decoder.mp3 role has been set on this component implementation it obeys the definition of the audio\_decoder.mp3 standard component. It shall, for example, expose the defined audio input and output ports and support the mandated configs and parameters on those ports. Upon setting the role, the component shall populate the default values on all parameters, configs and internal state, according to the role.

Via the mechanisms defined below, the core extracts information about which roles are supported by which component implementation and, using this information, provides two convenient functions for the IL client to query about such support. Furthermore, a component implementation allows an IL client to set the role which defines its behavior.

#### **8.1.1 ComponentRoleEnum**

The ComponentRoleEnum component function allows the IL core to query a component for all the roles it supports. This function allows the IL core to service OMX\_ComponentOfRoleEnum and OMX\_RoleOfComponentEnum calls. An efficient IL core will likely cache the role information it extracts from components (e.g. at installation) to avoid instantiating a component during OMX\_ComponentOfRoleEnum and OMX\_RoleOfComponentEnum calls.

ComponentRoleEnum enumerates (one role at a time) the component roles that a component supports.

```
OMX_ERRORTYPE (*ComponentRoleEnum)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_OUT OMX_U8* cRole,  
    OMX_IN U32 nIndex);
```

Parameters include:

- hComponent : The handle of the component that executes the call
- cRole: The name of the specified role. The role name string has a limit of 128 bytes (including ‘\0’).

- nIndex: The index of the role being queried.

### **8.1.2 OMX PARAM\_COMPONENTROLETYP**

The OMX\_PARAM\_COMPONENTROLETYP structure enables the IL client to set the role of the component via the OMX\_IndexParamStandardComponentRoleIndex. The result of a query on this parameter is undefined.

Setting this parameter on a component shall result in the component populating all the component data structures with default values according to the specified role. This includes defaults on all mandatory component and port parameters, and any other additional component structures.

All components shall support setting this parameter with the role name “default”. When this value is used, the component shall set all parameters, all configs and internal state as they are, when the component is first instantiated. The “default” role shall not be reported when the component is interrogated for supported roles.

```
typedef struct OMX_PARAM_COMPONENTROLETYP {
    OMX_U32 nSize;
    OMX_VERSONTYP;
    OMX_U8 cRole[OMX_MAX_STRINGNAME_SIZE];
}OMX_PARAM_COMPONENTROLETYP;
```

Parameters include:

- cRole: name of the role (i.e. name of the component definition).  
OMX\_MAX\_STRINGNAME\_SIZE is defined to have a value of 128.

### **8.1.3 OMX RoleOfComponentEnum**

The function that enables the IL client to query all the roles fulfilled by a given a component.

```
OMX_API OMX_ERRORTYPE OMX_APINTRY OMX_RoleOfComponentEnum (
    OMX_OUT OMX_STRING role,
    OMX_IN OMX_STRING compName,
    OMX_IN OMX_32 nIndex);
```

Parameters include:

- role: This is the role being returned. The caller shall provide a pointer to a valid string of at least 128 characters. If the function returns OMX\_ErrorNone then this string is filled in with the appropriate role of the component.
- compName: This is the name of the component being queried about.
- nIndex: A number containing the enumeration index for the role of the component. Multiple calls to OMX\_RoleOfComponentEnum with increasing values of nIndex will enumerate through the roles of the component until OMX\_ErrorNoMore is returned.

#### **8.1.4 OMX\_ComponentOfRoleEnum**

The OMX\_ComponentOfRoleEnum function that enables the IL client to query the names of all installed components that support a given role.

```
OMX_ERRORTYPE OMX_ComponentOfRoleEnum (
    OMX_OUT OMX_STRING compName,
    OMX_IN OMX_STRING role,
    OMX_IN OMX_U32 nIndex);
```

Parameters include:

- compName: The name of the component being returned. The caller shall provide a pointer to a valid string of at least 128 characters. If the function returns OMX\_ErrorNone then this string is filled in with the name of the appropriate component.
- role: This name of the role being queried about.
- nIndex: A number containing the enumeration index for the component implementing this role. Multiple calls to OMX\_ComponentOfRoleEnum with increasing values of nIndex will enumerate through the components implementing this role until OMX\_ErrorNoMore is returned.

## **8.2 Mandatory Port Parameters**

Across all standard components, OpenMAX IL 1.2 mandates support for certain parameters. Specifically:

- All standard components shall support the following parameters:
  - OMX\_IndexParamPortDefinition
  - OMX\_IndexParamCompBufferSupplier
  - OMX\_IndexParamAudioInit
  - OMX\_IndexParamImageInit
  - OMX\_IndexParamVideoInit
  - OMX\_IndexParamOtherInit
  - OMX\_IndexParamStandardComponentRole
- All *audio* ports on a standard component shall support the following parameters:
  - OMX\_IndexParamAudioPortFormat
- All *video* ports on a standard component shall support the following parameters:
  - OMX\_IndexParamVideoPortFormat
- All *image* ports on a standard component shall support the following parameters:
  - OMX\_IndexParamImagePortFormat
- All *other* ports on a standard component shall support the following parameters:

[o OMX\\_IndexParamOtherPortFormat](#)

## **89 Standard Components**

In the interest of facilitating strict component portability, OpenMAX IL defines a set of standard components. Each standard component definition associates specific interface criteria and functionality to the named standard component. To the extent these definitions are adhered to by clients and components, this allows one IL client to operate seamlessly with component implementations from multiple vendors and allows one component to operate seamlessly across multiple IL clients.

This section defines the set of OpenMAX IL standard components including:

- The hierarchy of standard component definitions.
- The mechanism for exposing standard components to an IL client.
- The definition of all standard classes and standard components.

### **8.19.1 Hierarchy of Standard Component Definition**

OpenMAX IL establishes two constructs for the hierarchical definition of the set of standard components:

- Standard component class: a category of standard components that share the same ports and high level functionality.
- Standard component: an instance of a standard component class that has the same ports and high level functionality as the class but that specifies the supported formats, parameters, and configs on those ports as well as the specific functionality of the component.

Thus OpenMAX IL divides the set of all standard components into classes of similar components, formally defining the characteristics of each class in terms of the ports it exposes and its overall function. Within each class, OpenMAX IL identifies specific standard components, formally defining the formats, parameters, and config operations supported on each port as well the specific type of functionality the individual component supports.

For instance, OpenMAX IL defines an `audio_decoder` class that represents all components that receive encoded audio on a single audio input port and emit decoded audio on single audio output. Furthermore, the `audio_decoder` class contains a standard component definition for each audio format: `audio_decoder.aac`, `audio_decoder.amr`, `audio_decoder.amr`, etc.

The difference in functionality between components in the previous example is the specific format of audio decoding implemented. However, the differences between components in a single class may also be distinguished in terms of their specific functionality. Each component in the `audio_processing` class, for example, operates on the same format (i.e. `pcm` audio) but implements different effects, e.g. `audio_processing.pcm.stereo_widening_loudspeakers`.

Thus, generally speaking, a component class defines a category of functionality and each component in that class implements one specific type of functionality within that category.

#### **8.1.19.1.1 Standard Component Class Definition**

The definition of a standard component class consists of:

- *Name*: The name of the standard component class.
- *Description*: Description of high level functionality.
- The set of ports exposed including the following information for each port:
  - *Index*: the index of the port.
  - *Domain*: the port's domain (audio, video, image, or other).
  - *Direction*: the ports direction (input or output).
  - *Description*: a description of the port's functionality relative to the component.

#### **8.1.29.1.2 Standard Components Definition**

The definition of a standard component consists of:

- *Name*: The name of the standard component.
- *Description*: Description of the specific functionality implemented by the component.
- For each port:
  - *Index*: The index of the described port.
  - *Description*: Description of the functionality implemented by the port relative to the component.
  - *Parameters and Configs*: A list of supported OpenMAX IL parameters and configs including the following information for each.
    - *Index*: The index value of the parameter or config used from the OMX\_INDEXTYPE enumeration.
    - *Access*: The read/write access of the parameter/config which is a any combination of the following:
      - *Read*: IL client is querying a component value via `GetParameter` or `GetConfig`. The component will fill in the appropriate fields of the structure passed.
      - *Write*: IL client is setting a component value via `SetParameter` or `SetConfig`. The IL client will fill in the appropriate fields of the structure passed.

- *Description*: Description of the parameter or config's function relative to the port.

## 8.2 Component Role

A component implementation may support one or more roles. We define a role as the behavior of component acting according to a particular standard component definition. The name of the standard component defining the behavior identifies the role.

For example a given component implementation named "OMX.CompanyXYZ.MyAudioDecoder" might support the following roles:

```
-----audio_decoder.mp3
-----audio_decoder.aac
-----audio_decoder.amr
```

When this component implementation is in the audio\_decoder.mp3 role it obeys the definition of the audio\_decoder.mp3 standard component. It shall, for example, expose the defined audio input and output ports, support the mandated configs and parameter on those ports, and populate the mandated defaults on those configs and parameters.

Via the mechanisms defined below, the core extracts information about which roles are supported by which component implementation and, using this information, provides two convenient functions for the IL client to query about such support. Furthermore, a component implementation allows an IL client to select the role which defines its behavior.

### 8.2.1 ComponentRoleEnum

The ComponentRoleEnum component function allows the IL core to query a component for all the roles it supports. This function allows the IL Core to service OMX\_GetComponentsOfRole and OMX\_GetRolesOfComponent calls. An efficient IL Core will likely cache the role information it extracts from components (e.g. at installation) to avoid instantiating a component during OMX\_GetComponentsOfRole and OMX\_GetRolesOfComponent calls.

ComponentRoleEnum enumerates (one role at a time) the component roles that a component supports.

```
OMX_ERRORTYPE (*ComponentRoleEnum)(  
    OMX_IN OMX_HANDLETYPE hComponent,  
    OMX_OUT OMX_STRING eRole,  
    OMX_IN U32 nIndex);
```

Parameters include:

- **hComponent**: The handle of the component that executes the call
- **eRole**: The name of the specified role. The role name string has a limit of 128 bytes (including '\0').

- **nIndex:** The index of the role being queried.

### **8.2.2 OMX\_PARAM\_COMPONENTROLETYPE**

The OMX\_PARAM\_COMPONENTROLETYPE structure represents the current role of the component that may be queried and set via the OMX\_IndexParamStandardComponentRole parameter. This enables the IL client to set the role of the component. The component populates defaults according to the specified role:

```
typedef struct OMX_PARAM_COMPONENTROLETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U8 eRole[OMX_MAX_STRINGNAME_SIZE];
}OMX_PARAM_COMPONENTROLETYPE;
```

Parameters include:

- **eRole:** name of the role (i.e. name of the standard component defining current behavior).
- OMX\_MAX\_STRINGNAME\_SIZE is defined to have a value of 128.

### **8.2.3 OMX\_GetRolesOfComponent**

The function that enables the IL client to query all the roles fulfilled by a given a component.

```
OMX_ERRORTYPE OMX_GetRolesOfComponent (
    OMX_STRING compName,
    OMX_U32 *pNumRoles,
    OMX_U8 **roles);
```

Parameters include:

- **compName:** This is the name of the component being queried about.
- **pNumRoles:** This is used both as input and output. On input it bounds the size of the input structure. On output it specifies how many roles were retrieved.
- **roles:** This is a list of the names of all standard components implemented on the specified physical component name. If this pointer is NULL this function populates the pNumRoles field with the number of roles the component supports and ignores the roles field. This allows the client to properly size the roles array on a subsequent call.

### **8.2.4 OMX\_GetComponentsOfRole**

The OMX\_GetComponentsOfRole function that enables the IL client to query the names of all installed components that support a given role.

```
OMX_ERRORTYPE OMX_GetComponentsOfRole (
    OMX_STRING role,
```

```
OMX_U32 *pNumComps,  
OMX_STRING **compNames);
```

Parameters include:

- **role**: The name of the specified role.
- **pNumComps**: This is used both as input and output. On input it bounds the size of the input structure. On output it specifies how many names were retrieved.
- **compNames**: This is a list of the names of all physical components that implement the specified standard component name. If this pointer is **NULL** this function populates the **pNumComps** field with the number of components that support the given role and ignores the **compNames** field. This allows the client to properly size the **compNames** field on a subsequent call.

### 8.3 Mandatory Port Parameters

Across all standard components, OpenMAX IL 1.1 mandates support for certain parameters. Specifically:

- All standard components shall support the following parameters:
  - OMX\_IndexParamPortDefinition
  - OMX\_IndexParamCompBufferSupplier
  - OMX\_IndexParamAudioInit
  - OMX\_IndexParamImageInit
  - OMX\_IndexParamVideoInit
  - OMX\_IndexParamOtherInit
- All *audio* ports on a standard component shall support the following parameters:
  - OMX\_IndexParamAudioPortFormat
- All *video* ports on a standard component shall support the following parameters
  - OMX\_IndexParamVideoPortFormat
- All *image* ports on a standard component shall support the following parameters:
  - OMX\_IndexParamImagePortFormat
- All *other* ports on a standard component shall support the following parameters:
  - OMX\_IndexParamOtherPortFormat

These requirements apply to all component described in this section though, for the sake of brevity, they have not been repeated for each standard class and component specification.

### **8.49.2 Notation Used**

The standard component definitions use certain conventions in their notation. Specifically:

- “APB” denotes the audio port base which is defined to be the nStartPortNumber value returned on a query of the OMX\_IndexParamAudioInit param.
- “IPB” denotes the image audio port base which is defined to be the nStartPortNumber value returned on a query of the OMX\_IndexParamImageInit param.
- “VPB” denotes the video port base which is defined to be the nStartPortNumber value returned on a query of the OMX\_IndexParamVideoInit param.
- “OPB” denotes the other port base which is defined to be the nStartPortNumber value returned on a query of the OMX\_IndexParamOtherInit param.

Furthermore, when a field of a parameter or config is specified all the listed values in the ‘Description’ column shall be supported and the *italicized* value shall be the default. A component that supports multiple standard component roles shall populate its fields with default settings according to the current role.

All parameter and config settings specified indicate the minimum settings that the components shall support to be categorized as a standard components.

### **8.59.3 Video and Image Order of Operations**

As part of the Video and Image domain, features have been defined that will apply data transform operations to data payloads. These data transforms consist of cropping, rotation, mirroring and scaling.

Depending on the ordering of the transforms applied to the data payload varying results will be produced. In order for the [IL client](#) to deterministically achieve a desired output among standard components that support such operations, the order of the these transforms applied to the data payload on a per port basis shall be as follows:

1. Cropping
2. Rotation
3. Mirroring
4. Scaling

This order is to be applied by components that support all or a subset of transforms.

For example:

- If a port within standard component A supports all four transforms then the order will be cropping followed by rotation followed by mirroring followed by scaling

- If a port within standard component B supports just three of the transforms – cropping, rotation and scaling – then the order will be cropping followed by rotation followed by scaling

Implementations of standard components supporting these transforms are not required to internally implement these transforms as outlined, rather the standard component implementations need to apply the operations to the payload in the logical order outlined such that a deterministic output is achieved.

This ordering of operations provides consistency for the IL client between different standard component implementations. It does not dictate the implementation of those components.

## **8.69.4 Standard Audio Components**

### **8.6.19.4.1 Audio Decoder Class**

<b>Name</b>	audio_decoder			
<b>Description</b>	Decodes the given compressed audio stream into an uncompressed audio stream.			
<b>Ports</b>	<b>Index</b> <b>Domain</b> <b>Direction</b> <b>Description</b>			
	APB+0	audio	input	Accepts encoded audio.
	APB+1	audio	output	Emits decoded audio.

Sample rate conversions, downmix and upmix support are not mandated. If these features are not supported, the component shall implement the slaving behavior as described in section (ref to slaving behavior section).

### **8.6.1.19.4.1.1 AAC Decoder Component**

<b>Name</b>	audio_decoder.aac			
<b>Description</b>	Decodes the given compressed audio stream into an uncompressed audio stream.			
<b>Ports</b>	<b>Index</b> <b>Domain</b> <b>Direction</b> <b>Description</b>			
	APB+0	audio	input	Accepts encoded audio.
	APB+1	audio	output	Emits decoded audio.

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts encoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b> OMX_IndexParamPortDefinition	<b>Access</b> r/w	<b>Description</b> Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingAAC
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingAAC

<b>Port Index</b>	APB+0		
	OMX_IndexParamAudioAac	r/w	<p>nChannels =  <i>2 (stereo)</i>  <i>1 (mono)</i></p> <p>nSampleRate =  8000  11025  12000  16000  22050  24000  32000  44100  48000</p> <p>nBitRate = up to 288Kbps per channel</p> <p>eAACProfile =  <i>OMX_AUDIO_AACObjectLC</i>  <i>OMX_AUDIO_AACObjectHE</i>  <i>OMX_AUDIO_AACObjectHE_PS</i></p> <p>eAACStreamFormat =  <i>OMX_AUDIO_AACStreamFormatMP2A</i>  <i>DTS</i>  <i>OMX_AUDIO_AACStreamFormatMP4</i>  <i>ADTS</i>  <i>OMX_AUDIO_AACStreamFormatADI</i>  <i>F</i>  <i>OMX_AUDIO_AACStreamFormatRA</i>  <i>W (headerless)</i></p> <p>eChannelMode =  <i>OMX_AUDIO_ChannelModeStereo</i>  <i>OMX_AUDIO_ChannelModeMono</i></p>

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits decoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = <i>OMX_AUDIO_CodingPCM</i>
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = <i>OMX_AUDIO_CodingPCM</i>

Port Index	APB+1			
	OMX_IndexParamAudioPcm	r/w	<p>nChannels = 2 (stereo) 1 (mono)</p> <p>eNumData = <i>OMX_NumericalDataSigned</i></p> <p>nSampleRate = <del>48000</del> 8000 11025 12000 16000 22050 24000 32000 44100 48000</p> <p>ePCMMMode = <i>OMX_AUDIO_PCMMModeLinear</i></p> <p>nBitPerSample = 16</p>	

#### 8.6.1.29.4.1.2 AMR-NB Decoder Component

Name	audio_decoder.amrnb			
Description	Decodes the given compressed audio stream into an uncompressed audio stream.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	input	Accepts encoded audio.
	APB+1	audio	output	Emits decoded audio.

Port Index	APB+0			
Description	Accepts encoded audio.			
Required Parameters/Configs	Index	Access	Description	
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = <i>OMX_AUDIO_CodingAMR</i>	
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = <i>OMX_AUDIO_CodingAMR</i>	

Port Index	APB+0		
	OMX_IndexParamAudioAmr	r/w	<p>nChannels = 1</p> <p>nBitRate =</p> <ul style="list-style-type: none"> <li>4750</li> <li>5150</li> <li>5900</li> <li>6700</li> <li>7400</li> <li>7950</li> <li>10200</li> <li>12200</li> </ul> <p>OMX_AUDIO_PARAM_AMRTYPE::</p> <p>OMX_AUDIO_AMRBANDMODETYPE =</p> <ul style="list-style-type: none"> <li>OMX_AUDIO_AMRBandModeNB0</li> <li>OMX_AUDIO_AMRBandModeNB1</li> <li>OMX_AUDIO_AMRBandModeNB2</li> <li>OMX_AUDIO_AMRBandModeNB3</li> <li>OMX_AUDIO_AMRBandModeNB4</li> <li>OMX_AUDIO_AMRBandModeNB5</li> <li>OMX_AUDIO_AMRBandModeNB6</li> <li>OMX_AUDIO_AMRBandModeNB7</li> </ul> <p>eAMRDTXMode =</p> <p>OMX_AUDIO_AMRDTXModeOnAuto</p> <p>#</p> <p>OMX_AUDIO_AMRDTXModeOnVA</p> <p>D1</p> <p>OMX_AUDIO_AMRDTXModeOnVA</p> <p>D2</p> <p>cAMRFrameFormat =</p> <p>OMX_AUDIO_AMRFrameFormatConf</p> <p>ormance</p> <p>OMX_AUDIO_AMRFrameFormatIF+</p> <p>OMX_AUDIO_AMRFrameFormatIF2</p> <p>OMX_AUDIO_AMRFrameFormatFSF</p> <p>OMX_AUDIO_AMRFrameFormatRTP</p> <p>Payload</p>

Port Index	APB+1		
Description	Emits decoded audio.		
Required Parameters/Configs	Index	Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the audio port settings.</p> <p>eEncoding =</p> <p>OMX_AUDIO_CodingPCM</p>

<b>Port Index</b>	APB+1		
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM  nChannels = 1 (mono)
	OMX_IndexParamAudioPcm	r/w	eNumData = OMX_NumericalDataSigned  nSampleRate = 8000  ePCMMMode = OMX_AUDIO_PCMMModeLinear  nBitPerSample = 16

#### 8.6.1.39.4.1.3 AMR-WB Decoder Component

<b>Name</b>	audio_decoder.amrwrb			
<b>Description</b>	Decodes the given compressed audio stream into an uncompressed audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	APB+0	audio	input	Accepts encoded audio.
	APB+1	audio	output	Emits decoded audio.

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts encoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings.  eEncoding = OMX_AUDIO_CodingAMR
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingAMR

Port Index	APB+0		
	OMX_IndexParamAudioAmr	r/w	<p>nChannels = 1</p> <p>nBitRate =</p> <ul style="list-style-type: none"> <li>6600</li> <li>8850</li> <li>12650</li> <li>14250</li> <li>15850</li> <li>18250</li> <li>19850</li> <li>23050</li> <li>23850</li> </ul> <p>OMX_AUDIO_PARAM_AMRTYPE::</p> <p>OMX_AUDIO_AMRBANDMODETYPE =</p> <ul style="list-style-type: none"> <li>OMX_AUDIO_AMRBandModeWB0</li> <li>OMX_AUDIO_AMRBandModeWB1</li> <li>OMX_AUDIO_AMRBandModeWB2</li> <li>OMX_AUDIO_AMRBandModeWB3</li> <li>OMX_AUDIO_AMRBandModeWB4</li> <li>OMX_AUDIO_AMRBandModeWB5</li> <li>OMX_AUDIO_AMRBandModeWB6</li> <li>OMX_AUDIO_AMRBandModeWB7</li> <li>OMX_AUDIO_AMRBandModeWB8</li> </ul> <p>eAMRDTXMode =</p> <p>OMX_AUDIO_AMRDTXMode<del>OnAutoOff</del></p> <p><del>OMX_AUDIO_AMRDTXModeOnVA</del></p> <p><del>D4</del></p> <p><del>OMX_AUDIO_AMRDTXModeOnVA</del></p> <p><del>D2</del></p> <p>cAMRFrameFormat =</p> <p><del>OMX_AUDIO_AMRFrameFormatConf</del></p> <p><del>ormance</del></p> <p><del>OMX_AUDIO_AMRFrameFormatIF+</del></p> <p><del>OMX_AUDIO_AMRFrameFormatIF2</del></p> <p><del>OMX_AUDIO_AMRFrameFormatFSF</del></p> <p><del>OMX_AUDIO_AMRFrameFormatRTP</del></p> <p><del>Payload</del></p>

Port Index	APB+1		
Description	Emits decoded audio.		
Required	Index	Access	Description

<b>Port Index</b>	APB+1		
<b>Parameters/ Configs</b>	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPcm	r/w	nChannels = 1 (mono)  eNumData = OMX_NumericalDataSigned  nSampleRate = 16000  ePCMMMode = OMX_AUDIO_PCMMModeLinear  nBitPerSample = 16

#### 9.4.1.4 AMR-WB+ Decoder Component

<b>Name</b>	audio_decoder.amrwb+		
<b>Description</b>	Decodes the given compressed audio stream into an uncompressed audio stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	APB+0	audio	Input
	APB+1	audio	output

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts encoded audio.		
<b>Required Parameters/ Configs</b>	<b>Index</b>	<b>Acce ss</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingAMR
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingAMR

<b>Port Index</b>	<b>APB+0</b>		
	<p><u>OMX_IndexParamAudioAmr</u></p> <p><u>r/w</u></p> <p><u>nChannels = 2 or 1</u></p> <p><u>nBitRate = between 5200 bps and 48000 bps</u></p> <p><u>OMX_AUDIO_PARAM_AMRTYPE::OMX_AUDIO_AMRBANDMODETYPE =</u></p> <p><u>Between</u></p> <p><u>    <u>OMX_AUDIO_AMRBandModeWBP0</u></u></p> <p><u>    <u>and</u></u></p> <p><u>    <u>OMX_AUDIO_AMRBandModeWBP47</u></u></p> <p><u>eAMRDTXMode =</u></p> <p><u>    <u>OMX_AUDIO_AMRDTXModeOnAuto</u></u></p> <p><u>eAMRFrameFormat =</u></p> <p><u>    <u>OMX_AUDIO_AMRFrameFormatWBPlusTIF</u></u></p> <p><u>    <u>OMX_AUDIO_AMRFrameFormatWBPlusSFSF</u></u></p> <p><u>eAMRISFIndex = between</u></p> <p><u>    <u>(OMX_AUDIO_AMRISFIndex0</u></u></p> <p><u>    <u>and</u></u></p> <p><u>    <u>OMX_AUDIO_AMRISFIndex13)</u></u></p> <p><u>    <u>or</u></u></p> <p><u>    <u>OMX_AUDIO_AMRISFIndexUnknown</u></u></p> <p><u>nSampleRate =</u></p> <p><u>    <u>48000</u></u></p> <p><u>    <u>44100</u></u></p> <p><u>    <u>32000</u></u></p> <p><u>    <u>24000</u></u></p> <p><u>    <u>22050</u></u></p> <p><u>    <u>16000</u></u></p> <p><u>    <u>11025</u></u></p> <p><u>    <u>8000</u></u></p>		

<b>Port Index</b>	<b>APB+1</b>
<b>Description</b>	<u>Emits decoded audio.</u>
<b>Required</b>	<u>Index</u>

<b>Port Index</b>	APB+1		
<b>Parameters/ Configs</b>			
	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<a href="#">Specify/query the audio port settings.</a> <a href="#">eEncoding = OMX_AUDIO_CodingPCM</a>
	<a href="#">OMX_IndexParamAudioPortFormat</a>	r/w	<a href="#">eEncoding = OMX_AUDIO_CodingPCM</a>
	<a href="#">OMX_IndexParamAudioPcm</a>	r/w	<a href="#">nChannels = 2 (stereo)</a> <a href="#">1 (mono)</a>  <a href="#">eNumData = OMX_NumericalDataSigned</a>  <a href="#">nSampleRate =</a> <a href="#">48000</a> <a href="#">44100</a> <a href="#">32000</a> <a href="#">24000</a> <a href="#">22050</a> <a href="#">16000</a> <a href="#">11025</a> <a href="#">8000</a>  <a href="#">ePCMMode = OMX_AUDIO_PCMModeLinear</a>  <a href="#">nBitPerSample = 16</a>

#### 8.6.1.49.4.1.5 MP3 Decoder Component

Name	audio_decoder.mp3			
Description	Decodes the given compressed audio stream into an uncompressed audio stream.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	input	Accepts encoded audio.
	APB+1	audio	output	Emits decoded audio.

Port Index	APB+0		
Description	Accepts encoded audio.		
Required Parameters/ Configs	Index	Access	Description
	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<a href="#">Specify/query the audio port settings.</a> <a href="#">eEncoding = OMX_AUDIO_CodingMP3</a>
	<a href="#">OMX_IndexParamAudioPortFormat</a>	r/w	<a href="#">eEncoding = OMX_AUDIO_CodingMP3</a>

<b>Port Index</b>	APB+0		
	OMX_IndexParamAudioMp3	r/w	<p>nChannels =  <i>2 (stereo)</i>  <i>1 (mono)</i></p> <p>nSampleRate =  <i>32000</i>  <i>44100</i>  <i>48000</i></p> <p>nBitRate =  <i>80000 to 320000</i></p> <p>eChannelMode =  <i>OMX_AUDIO_ChannelModeStereo</i>  <i>OMX_AUDIO_ChannelModeJointStereo</i>  <i>OMX_AUDIO_ChannelModeDual</i>  <i>OMX_AUDIO_ChannelModeMono</i></p> <p>eFormat =  <i>OMX_AUDIO_MP3StreamFormatMP1L</i>  <i>ayer3</i></p>

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits decoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the audio port settings.</p> <p>eEncoding =  <i>OMX_AUDIO_CodingPCM</i></p>
	OMX_IndexParamAudioPortFormat	r/w	<p>eEncoding =  <i>OMX_AUDIO_CodingPCM</i></p>

<b>Port Index</b>	APB+1		
	OMX_IndexParamAudioPcm	r/w	<p>nChannels =  <i>2 (stereo)</i>  <i>1 (mono)</i></p> <p>eNumData =  <i>OMX_NumericalDataSigned</i></p> <p>nSampleRate =  <i>32000</i>  <i>44100</i>  <i>48000</i></p> <p>ePCMMode =  <i>OMX_AUDIO_PCMMModeLinear</i></p> <p>nBitPerSample = <i>16</i></p>

#### **8.6.1.59.4.1.6 Real Audio Decoder Component**

<b>Name</b>	audio_decoder.ra			
<b>Description</b>	Decodes the given compressed audio stream into an uncompressed audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	APB+0	audio	input	Accepts encoded audio.
	APB+1	audio	output	Emits decoded audio.

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts encoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingRA
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingRA

<b>Port Index</b>	APB+0		
	OMX_IndexParamAudioRa	r/w	<p>nChannels =  <i>2 (stereo)</i>  <i>1 (mono)</i></p> <p>nBitRate =  <i>8000 to 96000 bps</i></p> <p>nSamplingRate =  <i>8000,</i>  <i>11025,</i>  <i>22050</i>  <i>44100</i></p> <p>nSample PerFrame = <i>256, 512, 1024</i></p> <p>eFormat =  <i>OMX_AUDIO_RA10_CODEC</i></p>

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits decoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the audio port settings.</p> <p>eEncoding =  <i>OMX_AUDIO_CodingPCM</i></p>
	OMX_IndexParamAudioPortFormat	r/w	<p>eEncoding =  <i>OMX_AUDIO_CodingPCM</i></p>
	OMX_IndexParamAudioPcm	r/w	<p>nChannels =  <i>2 (stereo)</i>  <i>1 (mono)</i></p> <p>eNumData =  <i>OMX_NumericalDataSigned</i></p> <p>nSampleRate =  <i>44100</i>  <i>8000</i>  <i>11025</i>  <i>22050</i>  <i>44100</i></p> <p>ePCMMMode =  <i>OMX_AUDIO_PCMMModeLinear</i></p> <p>nBitPerSample = <i>16</i></p>

### 8.6.1.69.4.1.7 WMA Decoder Component

Name	audio_decoder.wma			
Description	Decodes the given compressed audio stream into an uncompressed audio stream.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	input	Accepts encoded audio.
	APB+1	audio	output	Emits decoded audio.

Port Index	APB+0		
Description	Accepts encoded audio.		
Required Parameters/Configs	Index	Access	Description
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingWMA
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingWMA
	OMX_IndexParamAudioWma	r/w	nChannels = 2 (stereo) 1 (mono)  nBitRate = 5000 to 385000 bps  eFormat = OMX_AUDIO_WMAFormat9 OMX_AUDIO_WMAFormat8 OMX_AUDIO_WMAFormat7  nSamplingRate = 8000 11025 12000 16000 22050 24000 32000 44100 48000

Port Index	APB+1		
Description	Emits decoded audio.		
Required Parameters/Configs	Index	Access	Description
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM

Port Index	APB+1		
	OMX_IndexParamAudioPcm	r/w	<p>nChannels = 2 (stereo) 1 (mono)</p> <p>eNumData = <i>OMX_NumericalDataSigned</i></p> <p>nSampleRate = <b>48000</b> —8000 11025 12000 16000 22050 24000 32000 44100 48000</p> <p>ePCMMode = <i>OMX_AUDIO_PCMMModeLinear</i></p> <p>nBitPerSample = 16</p>

#### 8.6.29.4.2 Audio Encoder Class

Name	audio_encoder			
Description	Encodes the given audio stream into a compressed audio stream.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	input	Accepts audio for encoding.
	APB+1	audio	output	Emits encoded audio.

**Sample rate conversions, downmix and upmix support are not mandated. If these features are not supported, the component shall implement the slaving behavior as described in section (ref to slaving behavior section).**

#### 8.6.2.19.4.2.1 AAC Encoder Component

Name	audio_encoder.aac			
Description	Encodes the given audio stream into a compressed audio stream.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	input	Accepts audio for encoding.
	APB+1	audio	output	Emits encoded audio.

Port Index	APB+0
Description	Accepts audio for encoding.

<b>Port Index</b>	APB+0		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPcm	r/w	Specify/query the sampling rate and number of channels.  nChannels = 2 (Stereo) 1 (Mono)  eNumData = <i>OMX_NumericalDataSigned</i>  bInterleaved = <i>OMX_TRUE</i>  nBitPerSample = <i>16</i>  nSamplingRate = 8000 11025 12000 16000 22050 24000 32000 44100 48000  ePCMMMode = <i>OMX_AUDIO_PCMMModeLinear</i>

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits encoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingAAC
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingAAC

Port Index	APB+1		
	OMX_IndexParamAudioAac	r/w	<p>nChannels =  <i>2 (stereo)</i>  <i>1 (mono)</i></p> <p>nSampleRate =  <i>8000</i>  <i>11025</i>  <i>12000</i>  <i>16000</i>  <i>22050</i>  <i>24000</i>  <i>32000</i>  <i>44100</i>  <i>48000</i></p> <p>nBitRate = at least 288Kbps per channel</p> <p>nAudioBandWidth = 0</p> <p>nFrameLength = 0</p> <p>eAACProfile =  <i>OMX_AUDIO_AACObjectLC</i>  <i>OMX_AUDIO_AACObjectHE</i>  <i>OMX_AUDIO_AACObjectHE_PS</i></p> <p>eAACStreamFormat =  <i>OMX_AUDIO_AACStreamFormatMP2A</i>  <i>DTS</i>  <i>OMX_AUDIO_AACStreamFormatMP4</i>  <i>ADTS</i>  <i>OMX_AUDIO_AACStreamFormatADI</i>  <i>F</i>  <i>OMX_AUDIO_AACStreamFormatRA</i>  <i>W (headerless)</i></p> <p>eChannelMode =  <i>OMX_AUDIO_ChannelModeStereo</i>  <i>OMX_AUDIO_ChannelModeMono</i></p>

#### 8.6.2.29.4.2.2 AMR-NB Encoder Component

Name	audio_encoder.amrnrb			
Description	Encodes the given audio stream into a compressed audio stream.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	input	Accepts audio for encoding.
	APB+1	audio	output	Emits encoded audio.

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts audio for encoding.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPcm	r/w	Specify/query the sampling rate and number of channels.  nChannels = 1 (Mono)  eNumData = <i>OMX_NumericalDataSigned</i>  bInterleaved = <i>OMX_TRUE</i>  nBitPerSample = <i>16</i>  nSamplingRate = 8000  ePCMMMode = <i>OMX_AUDIO_PCMMModeLinear</i>

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits encoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingAMR
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingAMR

Port Index	APB+1		
	OMX_IndexParamAudioAmr	r/w	<p>nChannels = 1</p> <p>nBitRate =</p> <ul style="list-style-type: none"> <li>4750</li> <li>5150</li> <li>5900</li> <li>6700</li> <li>7400</li> <li>7950</li> <li>10200</li> <li>12200</li> </ul> <p>OMX_AUDIO_PARAM_AMRTYPE::</p> <p>OMX_AUDIO_AMRBANDMODETYPE =</p> <ul style="list-style-type: none"> <li>OMX_AUDIO_AMRBandModeNB0</li> <li>OMX_AUDIO_AMRBandModeNB1</li> <li>OMX_AUDIO_AMRBandModeNB2</li> <li>OMX_AUDIO_AMRBandModeNB3</li> <li>OMX_AUDIO_AMRBandModeNB4</li> <li>OMX_AUDIO_AMRBandModeNB5</li> <li>OMX_AUDIO_AMRBandModeNB6</li> <li>OMX_AUDIO_AMRBandModeNB7</li> </ul> <p>eAMRDTXMode =</p> <ul style="list-style-type: none"> <li>OMX_AUDIO_AMRDTXModeOff</li> <li>OMX_AUDIO_AMRDTXModeOnVA D1</li> <li>OMX_AUDIO_AMRDTXModeOnVA D2</li> </ul> <p>eAMRFrameFormat =</p> <p><del>OMX_AUDIO_AMRFrameFormatConf ormanee</del></p> <p><del>OMX_AUDIO_AMRFrameFormatIF1</del></p> <ul style="list-style-type: none"> <li>OMX_AUDIO_AMRFrameFormatIF2</li> <li>OMX_AUDIO_AMRFrameFormatFSF</li> <li><del>OMX_AUDIO_AMRFrameFormatRTP Payload</del></li> <li><del>OMX_AUDIO_AMRFrameFormatRTP Payload</del></li> </ul>

Port Index	APB+1
	<p><a href="#">OMX_IndexConfigAudioAmrMode</a></p> <p>r/w</p> <p>nBitRate =</p> <p><a href="#">4750</a></p> <p><a href="#">5150</a></p> <p><a href="#">5900</a></p> <p><a href="#">6700</a></p> <p><a href="#">7400</a></p> <p><a href="#">7950</a></p> <p><a href="#">10200</a></p> <p><a href="#">12200</a></p> <p><a href="#">OMX_AUDIO_AMRMODETYPE::OMX_AUDIO_AMRBANDMODETYPE</a> =</p> <p><a href="#">OMX_AUDIO_AMRBandModeNB0</a></p> <p><a href="#">OMX_AUDIO_AMRBandModeNB1</a></p> <p><a href="#">OMX_AUDIO_AMRBandModeNB2</a></p> <p><a href="#">OMX_AUDIO_AMRBandModeNB3</a></p> <p><a href="#">OMX_AUDIO_AMRBandModeNB4</a></p> <p><a href="#">OMX_AUDIO_AMRBandModeNB5</a></p> <p><a href="#">OMX_AUDIO_AMRBandModeNB6</a></p> <p><a href="#">OMX_AUDIO_AMRBandModeNB7</a></p>

#### 8.6.2.39.4.2.3 AMR-WB Encoder Component

Name	audio_encoder.amrwrb			
Description	Encodes the given audio stream into a compressed audio stream.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	input	Accepts audio for encoding.
	APB+1	audio	output	Emits encoded audio.

Port Index	APB+0		
Description	Accepts audio for encoding.		
Required Parameters/Configs	Index	Access	Description
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM

<b>Port Index</b>	APB+0		
	OMX_IndexParamAudioPcm	r/w	<p>Specify/query the sampling rate and number of channels.</p> <p>nChannels = 1 (Mono)</p> <p>eNumData = <i>OMX_NumericalDataSigned</i></p> <p>bInterleaved = <i>OMX_TRUE</i></p> <p>nBitPerSample = <i>16</i></p> <p>nSamplingRate = 16000</p> <p>ePCMMode = <i>OMX_AUDIO_PCMModeLinear</i></p>

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits encoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the audio port settings.</p> <p>eEncoding = <i>OMX_AUDIO_CodingAMR</i></p>
	OMX_IndexParamAudioPortFormat	r/w	<p>eEncoding = <i>OMX_AUDIO_CodingAMR</i></p>

Port Index	APB+1		
	OMX_IndexParamAudioAmr	r/w	<p>nChannels = 1</p> <p>nBitRate =</p> <ul style="list-style-type: none"> <li>6600</li> <li>8850</li> <li>12650</li> <li>14250</li> <li>15850</li> <li>18250</li> <li>19850</li> <li>23050</li> <li>23850</li> </ul> <p>OMX_AUDIO_PARAM_AMRTYPE::</p> <p>OMX_AUDIO_AMRBANDMODETYPE =</p> <ul style="list-style-type: none"> <li>OMX_AUDIO_AMRBandModeWB0</li> <li>OMX_AUDIO_AMRBandModeWB1</li> <li>OMX_AUDIO_AMRBandModeWB2</li> <li>OMX_AUDIO_AMRBandModeWB3</li> <li>OMX_AUDIO_AMRBandModeWB4</li> <li>OMX_AUDIO_AMRBandModeWB5</li> <li>OMX_AUDIO_AMRBandModeWB6</li> <li>OMX_AUDIO_AMRBandModeWB7</li> <li>OMX_AUDIO_AMRBandModeWB8</li> </ul> <p>eAMRDTXMode =</p> <p>OMX_AUDIO_AMRDTXMode<u>OnAuto</u></p> <p><del>Off</del></p> <p><del>OMX_AUDIO_AMRDTXModeOnVA</del></p> <p><del>D4</del></p> <p><del>OMX_AUDIO_AMRDTXModeOnVA</del></p> <p><del>D2</del></p> <p>cAMRFrameFormat =</p> <p><del>OMX_AUDIO_AMRFrameFormatConf</del></p> <p><del>ormance</del></p> <p><del>OMX_AUDIO_AMRFrameFormatIF+</del></p> <p><del>OMX_AUDIO_AMRFrameFormatIF2</del></p> <p><del>OMX_AUDIO_AMRFrameFormatFSF</del></p> <p><del>OMX_AUDIO_AMRFrameFormatRTP</del></p> <p><del>Payload</del></p> <p><del>OMX_AUDIO_AMRFrameFormatRTP</del></p> <p><del>Payload</del></p>

<b>Port Index</b>	APB+1			
	<u><a href="#">OMX_IndexConfigAudioAmrMode</a></u>	r/w	<u><a href="#">nBitRate =</a></u> <u><a href="#">6600</a></u> <u><a href="#">8850</a></u> <u><a href="#">12650</a></u> <u><a href="#">14250</a></u> <u><a href="#">15850</a></u> <u><a href="#">18250</a></u> <u><a href="#">19850</a></u> <u><a href="#">23050</a></u> <u><a href="#">23850</a></u>	<u><a href="#">OMX_AUDIO_AMRMODETYPE::OMX_AUDIO_AMRBANDMODETYPE</a></u> = <u><a href="#">OMX_AUDIO_AMRBandModeWB0</a></u> <u><a href="#">OMX_AUDIO_AMRBandModeWB1</a></u> <u><a href="#">OMX_AUDIO_AMRBandModeWB2</a></u> <u><a href="#">OMX_AUDIO_AMRBandModeWB3</a></u> <u><a href="#">OMX_AUDIO_AMRBandModeWB4</a></u> <u><a href="#">OMX_AUDIO_AMRBandModeWB5</a></u> <u><a href="#">OMX_AUDIO_AMRBandModeWB6</a></u> <u><a href="#">OMX_AUDIO_AMRBandModeWB7</a></u> <u><a href="#">OMX_AUDIO_AMRBandModeWB8</a></u>

#### [9.4.2.4 AMR-WB+ Encoder Component](#)

<u><a href="#">Name</a></u>	<u><a href="#">audio_encoder.amrwb+</a></u>			
<u><a href="#">Description</a></u>	Encodes the given audio stream into a compressed audio stream.			
<u><a href="#">Ports</a></u>	<u><a href="#">Index</a></u>	<u><a href="#">Domain</a></u>	<u><a href="#">Direction</a></u>	<u><a href="#">Description</a></u>
	<u><a href="#">APB+0</a></u>	<u><a href="#">audio</a></u>	<u><a href="#">input</a></u>	Accepts audio for encoding.
	<u><a href="#">APB+1</a></u>	<u><a href="#">audio</a></u>	<u><a href="#">output</a></u>	Emits encoded audio.

<b>Port Index</b>	APB+0			
<u><a href="#">Description</a></u>	Accepts audio for encoding.			
<u><a href="#">Required Parameters/Configs</a></u>	<u><a href="#">Index</a></u>	<u><a href="#">Access</a></u>	<u><a href="#">Description</a></u>	
	<u><a href="#">OMX_IndexParamPortDefinition</a></u>	r/w	Specify/query the audio port settings. eEncoding = <u><a href="#">OMX_AUDIO_CodingPCM</a></u>	
	<u><a href="#">OMX_IndexParamAudioPortFormat</a></u>	r/w	eEncoding = <u><a href="#">OMX_AUDIO_CodingPCM</a></u>	

<b>Port Index</b>	<b>APB+0</b>		
	<a href="#">OMX_IndexParamAudioPcm</a>	r/w	<p><u>Specify/query the sampling rate and number of channels.</u></p> <p><u>nChannels = 2 (Stereo) or</u>  <u>1 (Mono)</u></p> <p><u>eNumData =</u>  <u><i>OMX_NumericalDataSigned</i></u></p> <p><u>bInterleaved = OMX_TRUE</u></p> <p><u>nBitPerSample = 16</u></p> <p><u>nSamplingRate =</u></p> <p><u>48000</u></p> <p><u>44100</u></p> <p><u>32000</u></p> <p><u>24000</u></p> <p><u>22050</u></p> <p><u>16000</u></p> <p><u>11025</u></p> <p><u>8000</u></p> <p><u>ePCMMode =</u>  <u><i>OMX_AUDIO_PCMModeLinear</i></u></p>

<b>Port Index</b>	<b>APB+1</b>		
<b>Description</b>	<u>Emits encoded audio.</u>		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<u>Specify/query the audio port settings.</u> <u>eEncoding = OMX_AUDIO_CodingAMR</u>
	<a href="#">OMX_IndexParamAudioPortFormat</a>	r/w	<u>eEncoding = OMX_AUDIO_CodingAMR</u>

<b>Port Index</b>	<b>APB+1</b>
	<p><u>OMX_IndexParamAudioAmr</u></p> <p><u>r/w</u></p> <p><u>nChannels = 2 or 1</u></p> <p><u>nBitRate = between 5200 bps and 48000 bps</u></p> <p><u>OMX_AUDIO_PARAM_AMRTYPE::OMX_AUDIO_AMRBANDMODETYPE = Between</u>  <u>(OMX_AUDIO_AMRBandModeWBP0</u>  <u>and</u>  <u>OMX_AUDIO_AMRBandModeWBP47)</u>  <u>Or</u>  <u>OMX_AUDIO_AMRBandModeAuto</u></p> <p><u>eAMRDTXMode =</u>  <u>OMX_AUDIO_AMRDTXModeOnAuto</u></p> <p><u>eAMRISFIndex = between</u>  <u>(OMX_AUDIO_AMRISFIndex0</u>  <u>and</u>  <u>OMX_AUDIO_AMRISFIndex13)</u>  <u>Or</u>  <u>OMX_AUDIO_AMRISFIndexAuto</u></p> <p><u>eAMRFrameFormat =</u>  <u>OMX_AUDIO_AMRFrameFormatWBPlusTIF</u>  <u>OMX_AUDIO_AMRFrameFormatWBPlusFSF</u></p>

#### 8.6.2.49.4.2.5 MP3 Encoder Component

<b>Name</b>	audio_encoder.mp3			
<b>Description</b>	Encodes the given audio stream into a compressed audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	APB+0	audio	input	Accepts audio for encoding.
	APB+1	audio	output	Emits encoded audio.

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts audio for encoding.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM

<b>Port Index</b>	APB+0		
	OMX_IndexParamAudioPcm	r/w	<p>Specify/query the sampling rate and number of channels.</p> <p>nChannels = 2 (Stereo) 1 (Mono)</p> <p>eNumData = <i>OMX_NumericalDataSigned</i></p> <p>bInterleaved = <i>OMX_TRUE</i></p> <p>nBitPerSample = <i>16</i></p> <p>nSamplingRate = 32000 44100 48000</p> <p>ePCMMode = <i>OMX_AUDIO_PCMModeLinear</i></p>

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits encoded audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the audio port settings.</p> <p>eEncoding = <i>OMX_AUDIO_CodingMP3</i></p>
	OMX_IndexParamAudioPortFormat	r/w	<p>eEncoding = <i>OMX_AUDIO_CodingMP3</i></p>

Port Index	APB+1			
	OMX_IndexParamAudioMp3	r/w	nChannels = 2 (stereo) 1 (mono)  nBitRate = 80000 to 320000 bps  nSampleRate = 32000 44100 48000  nAudioBandWidth = 0  eChannelMode = OMX_AUDIO_ChannelModeStereo OMX_AUDIO_ChannelModeJointStereo OMX_AUDIO_ChannelModeDual OMX_AUDIO_ChannelModeMono	

#### **8.6.39.4.3 Audio Mixer Class**

The PCM format endianness is left to be native, meaning it can be either big endian or little endian depending on the underlying hardware. Endianness conversions, if needed, are left outside the standard audio post processing components.

Name	audio_mixer			
Description	Accepts multiple (N) audio streams, mixes them into a single stream, and emits the resulting stream as output.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	output	Emits audio stream resulting from mixing.
	APB+1 to APB+N	audio	input	Accepts audio stream for mixing.

#### **8.6.3.19.4.3.1 PCM Mixer Component**

Name	audio_mixer.pcm			
Description	Performs mixing of multiple audio input channels to 1 audio output mixing.			
Ports	Index	Domain	Direction	Description
	APB+0	audio	output	Emits audio stream resulting from mixing.
	APB+1 to APB+N	audio	input	Accepts audio stream for mixing.

Port Index	APB+0

<b>Port Index</b>	APB+0		
<b>Description</b>	Emits audio stream resulting from mixing.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPcm	r/w	Specify/query the sampling rate and number of channels.  nChannels = 2 (Stereo) 1 (Mono)  eNumData = <i>OMX_NumericalDataSigned</i>  eEndian = « Native »  bInterleaved = <i>OMX_TRUE</i>  nBitPerSample = 16  nSamplingRate = 8000, 11025 12000 16000 22050 24000 32000 44100 48000  ePCMMMode = <i>OMX_AUDIO_PCMMModeLinear</i>  eChannelMapping[0]= <i>OMX_AUDIO_ChannelLF (stereo)</i> <i>OMX_AUDIO_ChannelCF (mono)</i> eChannelMapping[1]= <i>OMX_AUDIO_ChannelRF (stereo)</i>
	OMX_IndexConfigAudioVolume	r/w	bLinear = <i>OMX_FALSE</i> sVolume = <i>Configurable</i>
	OMX_IndexConfigAudioMute	r/w	bMute = <i>OMX_FALSE</i>  <i>OMX_TRUE</i>

<b>Port Index</b>	APB+1 to APB+N		
<b>Description</b>	Accepts audio for mixing.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPcm	r/w	Specify/query the sampling rate and number of channels.  nChannels = 2 (Stereo) 1 (Mono)  eNumData = <i>OMX_NumericalDataSigned</i>  eEndian = « Native »  bInterleaved = <i>OMX_TRUE</i>  nBitPerSample = <i>16</i>  nSamplingRate = 8000, 11025 12000 16000 22050 24000 32000 44100 48000  ePCMMode = <i>OMX_AUDIO_PCMMModeLinear</i>  eChannelMapping[0]= <i>OMX_AUDIO_ChannelLF</i> (stereo) <i>OMX_AUDIO_ChannelICF</i> (mono) eChannelMapping[1]= <i>OMX_AUDIO_ChannelRF</i> (stereo)
	OMX_IndexConfigAudioVolume	r/w	bLinear = <i>OMX_FALSE</i> sVolume = <i>Configurable</i>

<b>Port Index</b>	APB+1 to APB+N		
	OMX_IndexConfigAudioMute	r/w	bMute = <i>OMX_FALSE</i> <i>OMX_TRUE</i>

#### **8.6.49.4.4 Audio Reader Class**

<b>Name</b>	audio_reader			
<b>Description</b>	Reads an audio filestream and emits contained audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	output	Emits audio stream found in filestream.

#### **8.6.4.19.4.4.1 Binary Audio Reader Class**

<b>Name</b>	audio_reader.binary			
<b>Description</b>	Blindly reads any audio filestream (e.g. an MP3 file) irrespective of format and emits contained elementary audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	output	Emits audio stream found in filestream.

#### **8.6.59.4.5 Audio Renderer Class**

The PCM format endianness is left to be native, meaning it can be either big endian or little endian depending on the underlying hardware. Endianness conversions, if needed, are left outside the standard audio post processing components.

**Audio renderers SHALL support providing reference clock.**

<b>Name</b>	audio_renderer			
<b>Description</b>	Renders a given audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	input	Accepts audio for presentation.
	OPB+0	other/time	input	Accepts time updates

#### **8.6.5.19.4.5.1 PCM Renderer Component**

<b>Name</b>	audio_renderer.pcm			
<b>Description</b>	Renders a given audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	input	Accepts audio for presentation.
	OPB+0	other/time	input	Accepts time updates

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts audio for rendering.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the audio port settings. eEncoding = <i>OMX_AUDIO_CodingPCM</i>
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = <i>OMX_AUDIO_CodingPCM</i>

Port Index	APB+0		
	OMX_IndexParamAudioPcm	r/w	<p>Specify/query the sampling rate and number of channels.</p> <p>nChannels = 2 (Stereo) 1 (Mono)</p> <p>eNumData = <i>OMX_NumericalDataSigned</i></p> <p>eEndian = « Native »</p> <p>bInterleaved = <i>OMX_TRUE</i></p> <p>nBitPerSample = <i>16</i></p> <p>nSamplingRate = 8000 11025 12000 16000 22050 24000 32000 44100 48000</p> <p>ePCMMode = <i>OMX_AUDIO_PCMMODELINEAR</i></p> <p>eChannelMapping[0]= <i>OMX_AUDIO_ChannelLF (stereo)</i> <i>OMX_AUDIO_ChannelCF (mono)</i> eChannelMapping[1]= <i>OMX_AUDIO_ChannelRF (stereo)</i></p>
	OMX_IndexConfigAudioVolume	r/w	<p>bLinear = <i>OMX_FALSE</i> sVolume = <i>Configurable</i></p>
	OMX_IndexConfigAudioMute	r/w	<p>bMute = <i>OMX_FALSE</i> <i>OMX_TRUE</i></p>

Port Index	OPB+0
<b>Description</b>	Accepts media time updates. Provides mechanism for audio renderer component to query for media time. Audio renderer can provide the audio reference clock to the clock component which facilitates synchronization of other processing (e.g. video rendering) to audio rendering.

<b>Port Index</b>	OPB+0
<b>Required Parameters/Configs</b>	<b>Index</b> OMX_IndexConfigTimeRenderingDelay <b>Access</b> Read <b>Description</b> Queries the new rendering delay

<b>Port Index</b>	OPB+0
<b>Description</b>	Accepts media time updates. Provides mechanism for audio renderer component to query for media time. Audio renderer can provide the audio reference clock to the clock component which facilitates synchronization of other processing (e.g. video rendering) to audio rendering..

#### **8.6.69.4.6 Audio Writer Class**

<b>Name</b>	audio_writer			
<b>Description</b>	Writes given audio stream to an audio filestream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	input	Accepts audio stream to be written to the audio filestream.

#### **8.6.6.19.4.6.1 Binary Audio Writer Class**

<b>Name</b>	audio_writer.binary			
<b>Description</b>	Blindly writes given elementary audio stream to an audio filestream (e.g. an MP3 file) irrespective of format.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	input	Accepts audio stream to be written to the audio filestream.

#### **8.6.79.4.7 Audio Capturer Class**

The PCM format endianness is left to be native, meaning it can be either big endian or little endian depending on the underlying hardware. Endianness conversions, if needed, are left outside the standard audio post processing components.

Scheduler by definition SHALL support being slave in A/V synchronization. Scheduler may not be required to provide reference clock updates. Audio capture components shall be capable of providing reference clock updates.

<b>Name</b>	audio_capturer			
<b>Description</b>	Emits an audio stream from an audio source.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	output	Emits source's audio stream.
	OPB+0	other/time	input	Receives media time updates/provides access to clock component.

#### **8.6.7.19.4.7.1 PCM Audio Capturer**

<b>Name</b>	audio_capturer.pcm
-------------	--------------------

<b>Name</b>	audio_capturer.pcm			
<b>Description</b>	Emits an audio stream from an audio source.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	APB+0	audio	output	Emits source's audio stream.

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts audio for rendering.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamAudioPortFormat	r/w	Specify/query the sampling rate and number of channels. eEncoding = <i>OMX_AUDIO_CodingPCM</i>
	OMX_IndexParamPortDefinition	r/w	eEncoding = <i>OMX_AUDIO_CodingPCM</i>

Port Index	APB+0		
	OMX_IndexParamAudioPcm		<p>Specify/query the sampling rate and number of channels.</p> <p>nChannels = 2 (Stereo) 1 (Mono)</p> <p>eNumData = <i>OMX_NumericalDataSigned</i></p> <p>eEndian = « Native »</p> <p>bInterleaved = <i>OMX_TRUE</i></p> <p>nBitPerSample = <i>16</i></p> <p>nSamplingRate = 8000 11025 12000 16000 22050 24000 32000 44100 48000</p> <p>ePCMMode = <i>OMX_AUDIO_PCMMODELINEAR</i></p> <p>eChannelMapping[0]= <i>OMX_AUDIO_ChannelLF (stereo)</i> <i>OMX_AUDIO_ChannelCF (mono)</i> eChannelMapping[1]= <i>OMX_AUDIO_ChannelRF (stereo)</i></p>
	OMX_IndexConfigAudioVolume	r/w	<p>bLinear = <i>OMX_FALSE</i> sVolume = <i>Configurable</i></p>
	OMX_IndexConfigAudioMute	r/w	<p>bMute = <i>OMX_FALSE</i> <i>OMX_TRUE</i></p>

Port Index	OPB+0		
Description	Accepts media time updates. Provides mechanism for audio capturer component to query for media time. Audio capturer can provide the audio reference clock to the clock component which facilitates synchronization of other processing (e.g. video capture) to audio capture.		

#### **8.6.7.29.4.7.2      Audio Capture Use Case**

An IL client using an audio source to capture an audio stream may do so via the following steps:

1. Instantiate the audio source component and any co-operating components
2. Set audio source settings:
3. Set the desired characteristics of the captured audio stream (e.g. sampling rate, channels)
4. Set the gain via the volume/mute controls
5. Establish any necessary tunnels between the audio source component and other components (e.g. an audio encoder tunneling with the capture port).
6. Select the clock component's active reference clock. In a use case with audio capture this is normally the audio clock as provided by the audio capturer.
7. Transition all components to the OMX\_StateIdle state. Then transition the audio source component to the OMX\_StatePause state, and transition all other components to the OMX\_StateExecuting state. Although all other components are ready for capture, the audio source's output port is not yet emitting data.
8. To initiate capture transition the audio source component to the OMX\_StateExecuting state. If using a clock component start the clock component. The audio source component will begin emitting captured audio of the prescribed characteristics.
9. To terminate capture transition the audio source component to the OMX\_StatePause state. The audio source component will cease emitting captured audio.

#### **8.6.89.4.8    Audio Processor class**

<b>Name</b>	audio_processor			
<b>Description</b>	Processes a raw audio stream			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	APB+0	audio	input	Accepts raw audio.
	APB+1	audio	output	Emits raw audio

#### **8.6.8.19.4.8.1    Properties that apply to all audio processing components**

Sample rate conversions, downmix and upmix support is not mandated. If these features are not supported, the component shall implement the slaving behavior as described in section (ref to slaving behavior section).

Sample rate conversions are not mandated. When the sampling rate of the input port is changed, the output port sampling rate shall automatically change to the same value.

The PCM format endianness is left to be native, meaning it can be either big endian or little endian depending on the underlying hardware. Endianness conversions, if needed, are left outside the standard audio post processing components.

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	eDomain = OMX_PortDomainAudio format.eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r/w	eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPcm	r/w	nChannels = 2 (Stereo)  eNumData = OMX_NumericalDataSigned  eEndian = <native>  bInterleaved = True  nBitPerSample = 16  ePCMMode = OMX_AUDIO_PCMMModeLinear  eChannelMapping = OMX_AUDIO_ChannelLF, OMX_AUDIO_ChannelRF

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r	eDomain = OMX_PortDomainAudio format.eEncoding = OMX_AUDIO_CodingPCM
	OMX_IndexParamAudioPortFormat	r	eEncoding = OMX_AUDIO_CodingPCM

Port Index	APB+1	r	nChannels = 2 (Stereo)  eNumData = OMX_NumericalDataSigned  eEndian = <native>  bInterleaved = True  nBitPerSample = 16  ePCMMMode = OMX_AUDIO_PCMMModeLinear  eChannelMapping = OMX_AUDIO_ChannelLF, OMX_AUDIO_ChannelRF
	OMX_IndexParamAudioPcm		

#### 8.6.8.29.4.8.2 Stereo widening loudspeakers

Headphone and loudspeaker versions of this standard component are separated to better support multi-components and to allow vendors to implement just one of the two algorithm variations.

In case the implementation supports only one single value for the nStereoWidening field of the OMX\_AUDIO\_CONFIG\_STEREOWIDENINGTYPE structure, that value shall be 100, and the component shall always return 100 as the value for the field for all OMX\_GetConfig calls. See Section 4.1.49—

[OMX\\_AUDIO\\_CONFIG\\_STEREOWIDENINGTYPE](#)[OMX\\_AUDIO\\_CONFIG\\_STEREOWIDENINGTYPE](#)

Formatted: Courier New

<b>Name</b>	audio_processor.pcm.stereo_widening_loudspeakers		
<b>Description</b>	Adds stereo widening to a raw audio stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	APB+0	audio	input
	APB+1	audio	output
	Accepts raw audio.		
	Emits raw audio		

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamAudioPcm	r/w	nBitPerSample = 16 nSamplingRate = 16000, 22050, 24000, 32000, 44100, 48000 Hz

<b>Port Index</b>	APB+1
<b>Description</b>	Emits raw audio.

<b>Port Index</b>	APB+1		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexConfigAudioStereoWidening	r/w	bEnable = <i>False</i> , True eWideningType = OMX_AUDIO_StereoWideningLoudspeakers
	OMX_IndexParamAudioPcm	r	nBitPerSample = <i>16</i> nSamplingRate = 16000, 22050, 24000, 32000, 44100, 48000 Hz

#### 8.6.8.39.4.8.3 Stereo widening headphones

In case the implementation supports only one single value for the nStereoWidening field of the OMX\_AUDIO\_CONFIG\_STEREOWIDENINGTYPE structure, that value shall be 100, and the component shall always return 100 as the value for the field for all OMX\_GetConfig calls. See Section [4.1.49](#)—

~~OMX\_AUDIO\_CONFIG\_STEREOWIDENINGTYPE~~  
~~OMX\_AUDIO\_CONFIG\_STEREOWIDENINGTYPE~~

Formatted: Courier New

<b>Name</b>	audio_processor.pcm.stereo_widening_headphones			
<b>Description</b>	Adds stereo widening to a raw audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	input	Accepts raw audio.
	APB+1	audio	output	Emits raw audio

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamAudioPcm	r/w	nBitPerSample = <i>16</i> nSamplingRate = 16000, 22050, 24000, 32000, 44100, 48000 Hz

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexConfigAudioStereoWidening	r/w	bEnable = <i>False</i> , True eWideningType = OMX_AUDIO_StereoWideningHeadphones
	OMX_IndexParamAudioPcm	r	nBitPerSample = <i>16</i> nSamplingRate = 16000, 22050, 24000, 32000, 44100, 48000 Hz

#### 8.6.8.49.4.8.4 Reverberation

<b>Name</b>	audio_processor.pcm.reverberation		
<b>Description</b>	Adds reverberation to a raw audio stream.		

<b>Name</b>	audio_processor.pcm.reverberation				
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>		
	APB+0	audio	input	Accepts raw audio.	
		APB+1	audio	output	Emits raw audio

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamAudioPcm	r/w	nBitPerSample = 16 nSamplingRate = 44100, 48000 Hz

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexConfigAudioReverberation	r/w	bEnable = False, True
		r	nBitPerSample = 16 nSamplingRate = 44100, 48000 Hz

#### 8.6.8.59.4.8.5 Chorus

<b>Name</b>	audio_processor.pcm.chorus				
<b>Description</b>	Adds chorus to a raw audio stream.				
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>		
	APB+0	audio	input	Accepts raw audio.	
		APB+1	audio	output	Emits raw audio

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamAudioPcm	r/w	nBitPerSample = 16 nSamplingRate = 44100, 48000 Hz

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexConfigAudioChorus	r/w	bEnable = False, True
		r	nBitPerSample = 16 nSamplingRate = 44100, 48000 Hz

#### 8.6.8.69.4.8.6 Equalizer

Equalizer band count is encoded into the name for convenience, so that the [IL client](#) can choose the preferred equalizer, if multiple exists, without loading the components.

<b>Name</b>	audio_processor.pcm.equalizer			
<b>Description</b>	Does equalization on a raw audio stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	input	Accepts raw audio.
	APB+1	audio	output	Emits raw audio

<b>Port Index</b>	APB+0		
<b>Description</b>	Accepts raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamAudioPcm	r	nBitPerSample = 16 nSamplingRate = 44100, 48000 Hz

<b>Port Index</b>	APB+1		
<b>Description</b>	Emits raw audio.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexConfigAudioEqualizer	r/w	bEnable = False, True sBandLevel = [-1200, 1200]
	OMX_IndexParamAudioPcm	r	nBitPerSample = 16 nSamplingRate = 44100, 48000 Hz
	OMX_IndexConfigAudioLoudness	r/w	bLoudness = False, True
	OMX_IndexConfigAudioBass	r/w	bEnable = False, True nBass = [-100, 100]
	OMX_IndexConfigAudioTreble	r/w	bEnable = False, True nTreble = [-100, 100]

#### 9.4.9 Audio 3D Mixer Class

A 3D Audio Mixer component accepts multiple (N) audio streams, applies positional 3D processing, and mixes the streams into two streams, and emits the two resulting streams as output. The first output stream contains the direct path audio and the second output stream contains the room signal.

The PCM format endianness is *native*, meaning it can be either big endian or little endian depending on the underlying hardware. Endianness conversions, if needed, are left outside the standard component.

<b>Name</b>	audio_3D_mixer			
<b>Description</b>	Accepts multiple (N) audio streams, 3D mixes them into two streams, and emits the two resulting streams as output. The first output contains the direct path audio and the second output contains the room signal.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	APB+0	audio	output	Emits audio stream resulting from positional 3D mixing.

	<u>APB+1</u>	<u>audio</u>	<u>output</u>	<u>Emits audio stream for room effect purposes.</u>
	<u>APB+2 to APB+(N+1)</u>	<u>audio</u>	<u>input</u>	<u>Accepts audio stream for mixing.</u>

#### 9.4.9.1 PCM 3D Mixer Component

Please note that there are two variants of this standard component role: one for headphone rendering and another one for loudspeaker rendering. The variants only differ in the mandated output type.

<u><b>Role Name</b></u>	<u>audio_3D_mixer_pcm.[headphones loudspeakers]</u>		
<u><b>Description</b></u>	<u>Performs 3D mixing of multiple (N) audio input channels to 2 audio output mixes: main mix (direct path audio) and room mix (for reverberator).</u>		
<u><b>Ports</b></u>	<u><b>Index</b></u>	<u><b>Domain</b></u>	<u><b>Direction</b></u>
	<u>APB+0</u>	<u>audio</u>	<u>output</u>
	<u>APB+1</u>	<u>audio</u>	<u>output</u>
	<u>APB+2 to APB+(N+1)</u>	<u>audio</u>	<u>input</u>

<u><b>Port Index</b></u>	<u>None</u>		
<u><b>Description</b></u>	<u>These required configs affect the whole component</u>		
<u><b>Required Parameters/Configs</b></u>	<u><b>Index</b></u>	<u><b>Access</b></u>	<u><b>Description</b></u>
	<u>OMX_IndexConfigCommitMode</u>	<u>r/w</u>	<u>Set or query the commit mode for the 3D mixer.  bDeferred =   <u>OMX_FALSE</u>   <u>OMX_TRUE</u></u>
	<u>OMX_IndexConfigCommit</u>	<u>w</u>	<u>Commits the 3D settings if commit mode is deferred.</u>

<u><b>Port Index</b></u>	<u>APB+0</u>		
<u><b>Description</b></u>	<u>Emits audio stream resulting from positional 3D mixing. This is the main output containing direct path audio.</u>		
<u><b>Required Parameters/Configs</b></u>	<u><b>Index</b></u>	<u><b>Access</b></u>	<u><b>Description</b></u>
	<u>OMX_IndexParamPortDefinition</u>	<u>r/w</u>	<u>Set or query the audio port settings. eEncoding =   <u>OMX_AUDIO_CodingPCM</u></u>
	<u>OMX_IndexParamAudioPortFormat</u>	<u>r/w</u>	<u>eEncoding =   <u>OMX_AUDIO_CodingPCM</u></u>

	<a href="#">OMX_IndexParamAudioPcm</a>	r/w	<p><u>Set or query the sampling rate and number of channels.</u></p> <p>nChannels =   <u>2 (Stereo)</u></p> <p>eNumData =   <u>OMX_NumericalDataSigned</u></p> <p>eEndian = « Native »</p> <p>bInterleaved = <u>OMX_TRUE</u></p> <p>nBitPerSample = <u>16</u></p> <p>nSamplingRate =   <u>8000</u>,   <u>11025</u>   <u>12000</u>   <u>16000</u>   <u>22050</u>   <u>24000</u>   <u>32000</u>   <u>44100</u>   <u>48000</u></p> <p>ePCMMode =   <u>OMX_AUDIO_PCMModeLinear</u></p> <p>eChannelMapping[0]=   <u>OMX_AUDIO_ChannelLF</u></p> <p>eChannelMapping[1]=   <u>OMX_AUDIO_ChannelRF</u></p>
	<a href="#">OMX_IndexConfigAudioVolume</a>	r/w	<p>bLinear = <u>OMX_FALSE</u> sVolume = <u>Configurable</u></p>
	<a href="#">OMX_IndexConfigAudioMute</a>	r/w	<p>bMute = <u>OMX_FALSE</u> <u>OMX_TRUE</u></p>
	<a href="#">OMX_IndexConfigAudio3DOutput</a>	r/w	<p>e3DOoutputType = <u>OMX_AUDIO_3DOutput[Headphones/Outputspeakers]</u></p>

Port Index	APB+1		
Description	Emits audio stream for room effect purposes.		
Required	Index	Access	Description

<b>Parameters/ Configs</b>	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<p>Set or query the audio port settings.</p> <p>eEncoding = <a href="#">OMX_AUDIO_CodingPCM</a></p>
	<a href="#">OMX_IndexParamAudioPortFormat</a>	r/w	eEncoding = <a href="#">OMX_AUDIO_CodingPCM</a>
	<a href="#">OMX_IndexParamAudioPcm</a>	r/w	<p>Set or query the sampling rate and number of channels.</p> <p>nChannels =</p> <ul style="list-style-type: none"> <li><u>1 (Mono) (conditional: only if the corresponding change in the Reverb's input port is made)</u></li> <li><u>2 (Stereo)</u></li> </ul> <p>eNumData = <a href="#">OMX_NumericalDataSigned</a></p> <p>eEndian = « Native »</p> <p>bInterleaved = <a href="#">OMX_TRUE</a></p> <p>nBitPerSample = <a href="#">16</a></p> <p>nSamplingRate =</p> <ul style="list-style-type: none"> <li><u>8000</u></li> <li><u>11025</u></li> <li><u>12000</u></li> <li><u>16000</u></li> <li><u>22050</u></li> <li><u>24000</u></li> <li><u>32000</u></li> <li><u>44100</u></li> <li><u>48000</u></li> </ul> <p>ePCMMode = <a href="#">OMX_AUDIO_PCMModeLinear</a></p> <p>eChannelMapping[0] = <a href="#">OMX_AUDIO_ChannelLF</a></p> <p>eChannelMapping[1] = <a href="#">OMX_AUDIO_ChannelRF</a></p>
	<a href="#">OMX_IndexConfigAudioVolume</a>	r/w	<p>bLinear = <a href="#">OMX_FALSE</a></p> <p>sVolume = <a href="#">Configurable</a></p>
	<a href="#">OMX_IndexConfigAudioMute</a>	r/w	<p>bMute = <a href="#">OMX_FALSE</a></p> <p><a href="#">OMX_TRUE</a></p>

<b>Port Index</b>	APB+2 to APB+(N+1)		
<b>Description</b>	Accepts audio for mixing.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<p>Set or query the audio port settings.</p> <p>eEncoding = <a href="#">OMX_AUDIO_CodingPCM</a></p>
	<a href="#">OMX_IndexParamAudioPortFormat</a>	r/w	<p>eEncoding = <a href="#">OMX_AUDIO_CodingPCM</a></p>
	<a href="#">OMX_IndexParamAudioPcm</a>	r/w	<p>Set or query the sampling rate and number of channels.</p> <p>nChannels =  <a href="#">1 (Mono)</a>  <a href="#">2 (Stereo)</a></p> <p>eNumData = <a href="#">OMX_NumericalDataSigned</a></p> <p>eEndian = « Native »</p> <p>bInterleaved = <a href="#">OMX_TRUE</a></p> <p>nBitPerSample = <a href="#">16</a></p> <p>nSamplingRate =  <a href="#">8000</a>  <a href="#">11025</a>  <a href="#">12000</a>  <a href="#">16000</a>  <a href="#">22050</a>  <a href="#">24000</a>  <a href="#">32000</a>  <a href="#">44100</a>  <a href="#">48000</a></p> <p>ePCMMode = <a href="#">OMX_AUDIO_PCMModeLinear</a></p> <p>eChannelMapping[0]= <a href="#">OMX_AUDIO_ChannelLF</a></p> <p>eChannelMapping[1]= <a href="#">OMX_AUDIO_ChannelRF</a></p>

	<u><a href="#">OMX_IndexParamAudio3DDopplerMode</a></u>	r/w	<p>Set or query the enabled status of the Doppler effect</p> <p>bEnabled =</p> <ul style="list-style-type: none"> <li><u><a href="#">OMX_FALSE</a></u></li> <li><u><a href="#">OMX_TRUE</a></u></li> </ul>
	<u><a href="#">OMX_IndexConfigAudio3DLocation</a></u>	r/w	<p>Set or query the virtual location of the 3D sound source.</p> <p>nX = the whole S32 range (default = 0)  nY = the whole S32 range (default = 0)  nZ = the whole S32 range (default = 0)</p>
	<u><a href="#">OMX_IndexConfigAudio3DDopplerSettings</a></u>	r/w	<p>Set or query the Doppler settings of the 3D sound source.</p> <p>nSoundSpeed = the whole U32 range (default = 340000)  nSourceVelocity = the whole S32 range (default = 0)  nListenerVelocity = the whole S32 range (default = 0)</p>
	<u><a href="#">OMX_IndexConfigAudio3DLevels</a></u>	r/w	<p>Set or query the direct path and room levels for the 3D sound source.</p> <p>sDirectLevel = [0x80000000OMX_MIN_S16, 0]  (default = 0)</p> <p>sRoomLevel = [0x80000000OMX_MIN_S16, 0]  (default = 0)</p>

	<u><a href="#">OMX_IndexConfigAudio3DDistanceAttenuation</a></u>	r/w	<p><u>Set or query the distance attenuation behavior for the 3D sound source.</u></p> <p><u>sMinDistance = [0, 0x7FFFFFFFOMX_MAX_S32] (default = 1000)</u></p> <p><u>sMaxDistance = [0, OMX_MAX_S32] (0x7FFFFFFFOMX_MAX_S32) (default = 0x7FFFFFFFOMX_MAX_S32)</u></p> <p><u>sRollOffFactor = [0, 10000] (default = 1000)</u></p> <p><u>sRoomRollOffFactor = [0, 10000] (default = 0)</u></p> <p><u>eRollOffModel =</u>  <u><i>OMX_AUDIO_RollOffExponential</i></u>  <u><i>OMX_AUDIO_RollOffLinear</i></u></p> <p><u>sMuteAfterMax =</u>  <u><i>OMX_FALSE</i></u>  <u><i>OMX_TRUE</i></u></p>
	<u><a href="#">OMX_IndexConfigAudio3DDirectivitySettings</a></u>	r/w	<p><u>Set or query the directivity behavior for the 3D sound source.</u></p> <p><u>sInnerAngle = [0, 360000] (default = 360000)</u></p> <p><u>sOuterAngle = [0, 360000] (default = 360000)</u></p> <p><u>sOuterLevel = [OMX_MIN_S160x80000000, 0] (default = 0)</u></p>
	<u><a href="#">OMX_IndexConfigAudio3DDirectivityOrientation</a></u>	r/w	<p><u>Set or query the orientation of the directivity of the 3D sound source.</u></p> <p><u>nXFront = the whole S32 range (default = 0)</u>  <u>nYFront = the whole S32 range (default = 0)</u>  <u>nZFront = the whole S32 range (default = -1000)</u></p>
	<u><a href="#">OMX_IndexConfigAudioMute</a></u>	r/w	<p><u>bMute =</u>  <u><i>OMX_FALSE</i></u>  <u><i>OMX_TRUE</i></u></p>

## 8.79.5 Standard Image Components

### 8.7.19.5.1 Image Decoder Class

<b>Name</b>	image_decoder			
<b>Description</b>	Decodes the given compressed image data stream into an uncompressed image data stream..			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	IPB+0	image	input	Accepts encoded image data.
	IPB+1	image	output	Emits decoded image data.

Upscaling and downscaling support is not mandated. If these features are not supported, the component shall implement the slaving behavior as described in section (ref to slaving behavior section).

### 8.7.1.19.5.1.1 JPEG Decoder

<b>Name</b>	image_decoder.JPGE			
<b>Description</b>	Decodes the given compressed image data stream into an uncompressed image data stream..			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	IPB+0	image	input	Accepts encoded image data.
	IPB+1	image	output	Emits decoded image data.

<b>Port Index</b>	IPB+0		
<b>Description</b>	Accepts encoded image data.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the image port settings. nFrameWidth = 640  nFrameHeight = 480  eCompressionFormat = <i>OMX_IMAGE_CodingJPEG</i>  eColorFormat = <i>OMX_COLOR_FormatUnused</i>
	OMX_IndexParamImagePortFormat	r/w	Specify/query the image format. eCompressionFormat = <i>OMX_IMAGE_CodingJPEG</i>  eColorFormat = <i>OMX_COLOR_FormatUnused</i>

<b>Port Index</b>	IPB+0		
	OMX_IndexParamQuantizationTable	r/w	eQuantizationTable= <i>OMX_IMAGE_QuantizationTableLuma</i> OMX_IMAGE_QuantizationTableChroma  nQuantizationMatrix = <i>configureable</i>
	OMX_IndexParamHuffmanTable	r/w	eHuffmanTable = <i>OMX_IMAGE_HuffmanTableAC</i> OMX_IMAGE_HuffmanTableDC  nNumberOfHuffmanCodeOfLength = <i>configurable</i>  nHuffmanTable = <i>configurable</i>

<b>Port Index</b>	IPB+1		
<b>Description</b>	Emits decoded image data.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the image port settings. nFrameWidth = 640</p> <p>nFrameHeight = 480</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><b>eColorFormat = At least one of the following:</b></p> <p><i>OMX_COLOR_FormatYUV420Planar</i>  <i>OMX_COLOR_FormatYUV420PackedPlanar</i>  <i>OMX_COLOR_FormatYUV420SemiPlanar</i>  <i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i>  <i>OMX_COLOR_FormatYVU420Planar</i>  <i>OMX_COLOR_FormatYVU420PackedPlanar</i>  <i>OMX_COLOR_FormatYVU420SemiPlanar</i>  <i>OMX_COLOR_FormatYVU420PackedSemiPlanar</i>  <b>eColorFormat =</b>  <i>OMX_COLOR_FormatYUV420Planar</i></p>

<b>Port Index</b>	IPB+1		
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the image format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><b>eColorFormat = At least one of the following:</b></p> <ul style="list-style-type: none"> <li><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></li> </ul> <p>eColorFormat = <u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

#### 9.5.1.1 WebP Decoder

<b>Name</b>	<u><a href="#">Image_decoder.WEBP</a></u>		
<b>Description</b>	Decodes the given compressed image data stream into an uncompressed image data stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	IPB+0	image	input
	IPB+1	image	output

<b>Port Index</b>	IPB+0		
<b>Description</b>	Accepts encoded image data.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the image port settings.</p> <p>nFrameWidth = 640</p> <p>nFrameHeight = 480</p> <p>eCompressionFormat = <u><a href="#">OMX_IMAGE_CodingWEBP</a></u></p> <p>eColorFormat = <u><a href="#">OMX_COLOR_FormatUnused</a></u></p>
	OMX_IndexParamImagePortFormat	r/w	<p>Specify/query the image format.</p> <p>eCompressionFormat = <u><a href="#">OMX_IMAGE_CodingWEBP</a></u></p> <p>eColorFormat =</p>

			<a href="#">OMX_COLOR_FormatUnused</a>								
<b>Port Index</b>	IPB+1										
<b>Description</b>	Emits decoded image data.										
<b>Required Parameters/Configs</b>	<table border="1"> <thead> <tr> <th><b>Index</b></th> <th><b>Access</b></th> <th><b>Description</b></th> </tr> </thead> <tbody> <tr> <td><a href="#">OMX_IndexParamPortDefinition</a></td> <td>r/w</td> <td> <p>Specify/query the image port settings.</p> <p><a href="#">nFrameWidth = 640</a></p> <p><a href="#">nFrameHeight = 480</a></p> <p><a href="#">eCompressionFormat = <i>OMX_IMAGE_CodingUnused</i></a></p> <p><a href="#">eColorFormat = At least one of the following:</a></p> <ul style="list-style-type: none"> <li><a href="#">OMX_COLOR_FormatYUV420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></li> </ul> </td> </tr> <tr> <td><a href="#">OMX_IndexParamVideoPortFormat</a></td> <td>r/w</td> <td> <p>Specify/query the image format.</p> <p><a href="#">eCompressionFormat = <i>OMX_IMAGE_CodingUnused</i></a></p> <p><a href="#">eColorFormat = At least one of the following:</a></p> <ul style="list-style-type: none"> <li><a href="#">OMX_COLOR_FormatYUV420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></li> </ul> </td> </tr> </tbody> </table>	<b>Index</b>	<b>Access</b>	<b>Description</b>	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<p>Specify/query the image port settings.</p> <p><a href="#">nFrameWidth = 640</a></p> <p><a href="#">nFrameHeight = 480</a></p> <p><a href="#">eCompressionFormat = <i>OMX_IMAGE_CodingUnused</i></a></p> <p><a href="#">eColorFormat = At least one of the following:</a></p> <ul style="list-style-type: none"> <li><a href="#">OMX_COLOR_FormatYUV420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></li> </ul>	<a href="#">OMX_IndexParamVideoPortFormat</a>	r/w	<p>Specify/query the image format.</p> <p><a href="#">eCompressionFormat = <i>OMX_IMAGE_CodingUnused</i></a></p> <p><a href="#">eColorFormat = At least one of the following:</a></p> <ul style="list-style-type: none"> <li><a href="#">OMX_COLOR_FormatYUV420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></li> </ul>	
<b>Index</b>	<b>Access</b>	<b>Description</b>									
<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<p>Specify/query the image port settings.</p> <p><a href="#">nFrameWidth = 640</a></p> <p><a href="#">nFrameHeight = 480</a></p> <p><a href="#">eCompressionFormat = <i>OMX_IMAGE_CodingUnused</i></a></p> <p><a href="#">eColorFormat = At least one of the following:</a></p> <ul style="list-style-type: none"> <li><a href="#">OMX_COLOR_FormatYUV420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></li> </ul>									
<a href="#">OMX_IndexParamVideoPortFormat</a>	r/w	<p>Specify/query the image format.</p> <p><a href="#">eCompressionFormat = <i>OMX_IMAGE_CodingUnused</i></a></p> <p><a href="#">eColorFormat = At least one of the following:</a></p> <ul style="list-style-type: none"> <li><a href="#">OMX_COLOR_FormatYUV420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420Planar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></li> <li><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></li> </ul>									

#### **8.7.29.5.2 Image Encoder Class**

<b>Name</b>	image_encoder			
<b>Description</b>	Encodes the given image data stream into a compressed format.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	IPB+0	image	input	Accepts image data for encoding.
	IPB+1	image	output	Emits compressed image data.

Upscaling and downscaling support is not mandated. If these features are not supported, the component shall implement the slaving behavior as described in section (ref to slaving behavior section).

#### **8.7.2.19.5.2.1 JPEG Encoder**

<b>Name</b>	image_encoder.JPEG			
<b>Description</b>	Encodes the given image data stream into a compressed format.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	IPB+0	image	input	Accepts image data for encoding.
	IPB+1	image	output	Emits compressed image data.

<b>Port Index</b>	IPB+0	
<b>Description</b>	Accepts image data for encoding.	
<b>Required</b>	<b>Index</b>	

Port Index	IPB+0		
Parameters/ Configs	Type	Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the image port settings.</p> <p>nFrameWidth = 640</p> <p>nFrameHeight = 480</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the image format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

<b>Port Index</b>	IPB+1		
<b>Description</b>	Emits compressed image data.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the image port settings.</p> <p>nFrameWidth = 640(<i>same as input</i>)</p> <p>nFrameHeight = 480(<i>same as input</i>)</p> <p>eCompressionFormat = <i>OMX_IMAGE_CodingJPEG</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamImagePortFormat	r/w	<p>Specify/query the image format.</p> <p>eCompressionFormat = <i>OMX_IMAGE_CodingJPEG</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamQuantizationTable	r/w	<p>eQuantizationTable= <i>OMX_IMAGE_QuantizationTableLuma</i>  <i>OMX_IMAGE_QuantizationTableChroma</i></p> <p>nQuantizationMatrix = <i>configureable</i></p>

### 9.5.2.1 WebP Encoder

<b>Name</b>	Image_encoder.WEBP		
<b>Description</b>	Encodes the given image data stream into a compressed format.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	IPB+0	image	input
	IPB+1	image	output

<b>Port Index</b>	IPB+0		
<b>Description</b>	Accepts image data for encoding.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the image port settings.</p> <p>nFrameWidth = 640</p> <p>nFrameHeight = 480</p> <p>eCompressionFormat = <i>OMX_IMAGE_CodingUnused</i></p> <p>eColorFormat = At least one of the</p>

			<p><u>following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p>
	<u><a href="#">OMX_IndexParamImagePortFormat</a></u>	r/w	<p><u>Specify/query the video format.</u>  <u>eCompressionFormat = <a href="#">OMX_IMAGE_CodingUnused</a></u></p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p>

<u><b>Port Index</b></u>	<u><a href="#">IPB+1</a></u>						
<u><b>Description</b></u>	<u>Emits compressed image data.</u>						
<u><b>Required Parameters/Configs</b></u>	<table border="1"> <tr> <th><u><b>Index</b></u></th> <th><u><b>Access</b></u></th> <th><u><b>Description</b></u></th> </tr> <tr> <td><u><a href="#">OMX_IndexParamPortDefinition</a></u></td> <td>r/w</td> <td> <p><u>Specify/query the image port settings.</u>  <u>nFrameWidth = 640 (same as input)</u></p> <p><u>nFrameHeight = 480 (same as input)</u></p> <p><u>eCompressionFormat = <a href="#">OMX_IMAGE_CodingWEBP</a></u></p> <p><u>eColorFormat = <a href="#">OMX_COLOR_FormatUnused</a></u></p> </td></tr> </table>	<u><b>Index</b></u>	<u><b>Access</b></u>	<u><b>Description</b></u>	<u><a href="#">OMX_IndexParamPortDefinition</a></u>	r/w	<p><u>Specify/query the image port settings.</u>  <u>nFrameWidth = 640 (same as input)</u></p> <p><u>nFrameHeight = 480 (same as input)</u></p> <p><u>eCompressionFormat = <a href="#">OMX_IMAGE_CodingWEBP</a></u></p> <p><u>eColorFormat = <a href="#">OMX_COLOR_FormatUnused</a></u></p>
<u><b>Index</b></u>	<u><b>Access</b></u>	<u><b>Description</b></u>					
<u><a href="#">OMX_IndexParamPortDefinition</a></u>	r/w	<p><u>Specify/query the image port settings.</u>  <u>nFrameWidth = 640 (same as input)</u></p> <p><u>nFrameHeight = 480 (same as input)</u></p> <p><u>eCompressionFormat = <a href="#">OMX_IMAGE_CodingWEBP</a></u></p> <p><u>eColorFormat = <a href="#">OMX_COLOR_FormatUnused</a></u></p>					

	<a href="#">OMX_IndexParamVideoPortFormat</a>	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = <a href="#">OMX IMAGE CodingWEBP</a></p> <p>eColorFormat = <a href="#">OMX COLOR FormatUnused</a></p>
--	---	-----	--

#### **8.7.39.5.3 Image Reader Class**

<b>Name</b>	image_reader			
<b>Description</b>	Read an image filestream and emits the contained image stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	IPB+0	image	output	Emits image stream found in filestream.

#### **8.7.3.19.5.3.1 Binary Image Reader Class**

<b>Name</b>	image_reader.binary			
<b>Description</b>	Blindly reads any image filestream (e.g. a JPG file) irrespective of the format and emits contained elementary image stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	IPB+0	image	output	Emits image stream found in filestream.

#### **8.7.49.5.4 Image Writer Class**

<b>Name</b>	image_writer			
<b>Description</b>	Writes given image stream to an image filestream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	IPB+0	image	input	Accepts image stream to be written to the image filestream.

#### **8.7.4.19.5.4.1 Binary Image Writer Class**

<b>Name</b>	image_writer.binary			
<b>Description</b>	Blindly writes given elementary image stream to an image filestream (e.g. a JPG file) irrespective of format.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	IPB+0	image	input	Accepts image stream to be written to the image filestream.

### **8.89.6 Standard Video Components**

#### **8.8.19.6.1 Video Decoder Class**

<b>Name</b>	video_decoder			
<b>Description</b>	Decodes the given compressed video stream into an uncompressed video stream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	
	VPB+0	video	input	Accepts encoded video.
	VPB+1	video	output	Emits decoded video.

Upscaling, downscaling and frame rate conversion support is not mandated. If these features are not supported, the component shall implement the slaving behavior as described in section (ref to slaving behavior section).

Video decoders shall emit frames in display order.

#### 8.8.1.19.6.1.1 H.263 Decoder Component

<b>Name</b>	video_decoder.h263		
<b>Description</b>	Decodes the given compressed video stream into an uncompressed video stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input
	VPB+1	video	output
			Consumes compressed video content.
			Produces uncompressed raw video.

<b>Port Index</b>	VPB+0		
<b>Description</b>	Consumes compressed video content.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the video port settings. nFrameWidth = 176  nFrameHeight = 144  nBitRate = 64000  xFrameRate = 15  eCompressionFormat = <i>OMX_VIDEO_CodingH263</i>  eColorFormat = <i>OMX_COLOR_FormatUnused</i>
	OMX_IndexParamVideoPortFormat	r/w	Specify/query the video format. eCompressionFormat = <i>OMX_VIDEO_CodingH263</i>  eColorFormat = <i>OMX_COLOR_FormatUnused</i>
	OMX_IndexParamVideoH263	r	eProfile = <i>OMX_VIDEO_H263ProfileBaseline</i>  eLevel= <i>OMX_VIDEO_H263Level10</i>  bPLUSPTYPEAllowed = <i>OMX_FALSE</i>  bForceRoundingTypeToZero = <i>OMX_TRUE</i>

Port Index	VPB+0		
	OMX_IndexParamVideoProfileLevel QuerySupported	r	Query supported profile/level pair by index.
	OMX_IndexParamVideoProfileLevel Current	r	Query current profile/level pair.

Port Index	VPB+1		
Description	Produces uncompressed raw video.		
Required Parameters/Configs	Index	Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><b>eColorFormat = At least one of the following:</b></p> <p><i>OMX_COLOR_FormatYUV420Planar</i></p> <p><i>OMX_COLOR_FormatYUV420PackedPlanar</i></p> <p><i>OMX_COLOR_FormatYUV420SemiPlanar</i></p> <p><i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i></p> <p><i>OMX_COLOR_FormatYVU420Planar</i></p> <p><i>OMX_COLOR_FormatYVU420PackedPlanar</i></p> <p><i>OMX_COLOR_FormatYVU420SemiPlanar</i></p> <p><i>OMX_COLOR_FormatYVU420PackedSemiPlanar</i></p> <p><b>eColorFormat =</b></p> <p><i>OMX_COLOR_FormatYUV420Planar</i></p>

Port Index	VPB+1		
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><b>eColorFormat = At least one of the following:</b></p> <ul style="list-style-type: none"> <li><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></li> </ul> <p>eColorFormat = <u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

#### 8.8.1.29.6.1.2 AVC Decoder Component

Name	video_decoder.avc			
Description	Decodes the given compressed video stream into an uncompressed video stream.			
Ports	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	VPB+0	video	input	Consumes compressed video content.
	VPB+1	video	output	Produces uncompressed raw video.

Port Index	VPB+0		
Description	Consumes compressed video content.		
Required	<b>Index</b>	<b>Access</b>	<b>Description</b>

Port Index	VPB+0		
Parameters/ Configs		Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>nBitRate = 64000</p> <p>xFrameRate = 15</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingAVC</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingAVC</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoAvc	r	<p>eProfile = <i>OMX_VIDEO_AVCPProfileBaseline</i></p> <p>eLevel = <i>OMX_VIDEO_AVCLevel1</i></p>
	OMX_IndexParamVideoProfileLevel QuerySupported	r	Query supported profile/level pair by index.
	OMX_IndexParamVideoProfileLevel Current	r	Query current profile/level pair.

Port Index	VPB+1		
Description	Produces uncompressed raw video.		
Required	Index	Access	Description

Port Index	VPB+1		
Parameters/ Configs	Type	Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

### 8.8.1.39.6.1.3 MPEG4 Video Decoder Component

<b>Name</b>	video_decoder.mpeg4		
<b>Description</b>	Decodes the given compressed video stream into an uncompressed video stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input
	VPB+1	video	output
			Produces uncompressed raw video.

<b>Port Index</b>	VPB+0		
<b>Description</b>	Consumes compressed video content.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the video port settings. nFrameWidth = 176 nFrameHeight = 144 nBitRate = 64000 xFrameRate = 15 eCompressionFormat = <i>OMX_VIDEO_CodingMPEG4</i> eColorFormat = <i>OMX_COLOR_FormatUnused</i>
	OMX_IndexParamVideoPortFormat	r/w	Specify/query the video format. eCompressionFormat = <i>OMX_VIDEO_CodingMPEG4</i> eColorFormat = <i>OMX_COLOR_FormatUnused</i>
	OMX_IndexParamVideoMpeg4	r/w	eProfile = <i>OMX_VIDEO_MPEG4ProfileSimple</i> eLevel = <i>OMX_VIDEO_MPEG4Level1</i>
	OMX_IndexParamVideoProfileLevelQuerySupported	r	Query supported profile/level pair by index.
	OMX_IndexParamVideoProfileLevelCurrent	r	Query current profile/level pair.

<b>Port Index</b>	VPB+1		
<b>Description</b>	Produces uncompressed raw video.		
<b>Required</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

Port Index	VPB+1		
Parameters/ Configs	Type	Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

#### 8.8.1.49.6.1.4 Real Video Decoder Component

<b>Name</b>	video_decoder.rv		
<b>Description</b>	Decodes the given compressed video stream into an uncompressed video stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input
	VPB+1	video	output
			Produces uncompressed raw video.

<b>Port Index</b>	VPB+0		
<b>Description</b>	Consumes compressed video content.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>nBitRate = 64000</p> <p>xFrameRate = 15</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingRV</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingRV</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoRv	r/w	<p>Specify/query Real Video specific parameters.</p> <p>eFormat = <i>OMX_VIDEO_RVFormat8</i> <i>OMX_VIDEO_RVFormat9</i></p> <p>bEnablePostFilter = <i>OMX_TRUE</i> <i>OMX_FALSE</i></p> <p>bEnableLatencyMode = <i>OMX_TRUE</i> <i>OMX_FALSE</i></p>

<b>Port Index</b>	VPB+1
<b>Description</b>	Produces uncompressed raw video.

<b>Port Index</b>	VPB+1		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

### 8.8.1.59.6.1.5 WMV Decoder Component

<b>Name</b>	video_decoder.wmv		
<b>Description</b>	Decodes the given compressed video stream into an uncompressed video stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input
	VPB+1	video	output
			Produces uncompressed raw video.

<b>Port Index</b>	VPB+0		
<b>Description</b>	Consumes compressed video content.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>nBitRate = 64000</p> <p>xFrameRate = 15</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingWMV</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingWMV</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoWmv	r/w	<p>Specify/query Real Video specific parameters.</p> <p>eFormat = <i>OMX_VIDEO_WMVFormat7</i></p> <p><i>OMX_VIDEO_WMVFormat8</i></p> <p><i>OMX_VIDEO_WMVFormat9</i></p>

<b>Port Index</b>	VPB+1		
<b>Description</b>	Produces uncompressed raw video.		
<b>Required</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

Port Index	VPB+1		
Parameters/ Configs	Type	Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

### **9.6.1.6 VC-1 Decoder Component**

<b>Name</b>	video_decoder.vc1		
<b>Description</b>	Decodes the given compressed video stream into an uncompressed video stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	Video	input
	VPB+1	Video	output

<b>Port Index</b>	VPB+0		
<b>Description</b>	Consumes compressed video content.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>nBitRate = 64000</p> <p>xFrameRate = 15</p> <p>eCompressionFormat = <a href="#">OMX_VIDEO_CodingVC1</a></p> <p>eColorFormat = <a href="#">OMX_COLOR_FormatUnused</a></p>
	<a href="#">OMX_IndexParamVideoPortFormat</a>	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = <a href="#">OMX_VIDEO_CodingVC1</a></p> <p>eColorFormat = <a href="#">OMX_COLOR_FormatUnused</a></p>
	<a href="#">OMX_IndexParamVideoVC1</a>	r/w	<p>Specify/query VC-1 specific parameters.</p> <p>eProfile = <a href="#">OMX_VIDEO_VC1ProfileSimple</a></p> <p><a href="#">OMX_VIDEO_VC1ProfileMain</a></p> <p><a href="#">OMX_VIDEO_VC1ProfileAdvanced</a></p>

<b>Port Index</b>	VPB+1		
<b>Description</b>	Produces uncompressed raw video.		
<b>Required</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

<b>Port Index</b>	VPB+1		
<b>Parameters/ Configs</b>	<u>OMX_IndexParamPortDefinition</u>	r/w	<p><u>Specify/query the video port settings.</u></p> <p><u>nFrameWidth = 176</u></p> <p><u>nFrameHeight = 144</u></p> <p><u>eCompressionFormat = OMX_VIDEO_CodingUnused</u></p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u>OMX_COLOR_FormatYUV420Planar</u></p> <p><u>OMX_COLOR_FormatYUV420PackedPlanar</u></p> <p><u>OMX_COLOR_FormatYUV420SemiPlanar</u></p> <p><u>OMX_COLOR_FormatYUV420PackedSemiPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420Planar</u></p> <p><u>OMX_COLOR_FormatYVU420PackedPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420SemiPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420PackedSemiPlanar</u></p>
	<u>OMX_IndexParamVideoPortFormat</u>	r/w	<p><u>Specify/query the video format.</u></p> <p><u>eCompressionFormat = OMX_VIDEO_CodingUnused</u></p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u>OMX_COLOR_FormatYUV420Planar</u></p> <p><u>OMX_COLOR_FormatYUV420PackedPlanar</u></p> <p><u>OMX_COLOR_FormatYUV420SemiPlanar</u></p> <p><u>OMX_COLOR_FormatYUV420PackedSemiPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420Planar</u></p> <p><u>OMX_COLOR_FormatYVU420PackedPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420SemiPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420PackedSemiPlanar</u></p>

### 9.6.1.1 VP8 Decoder Component

<b>Name</b>	video_decoder_vp8		
<b>Description</b>	Decodes the given compressed video stream into an uncompressed video stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input

<b>Port Index</b>	VPB+0		
<b>Description</b>	Consumes compressed video content.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<a href="#">Specify/query the video port settings.</a> <a href="#">nFrameWidth = 176</a>  <a href="#">nFrameHeight = 144</a>  <a href="#">nBitRate = 64000</a>  <a href="#">xFrameRate = 15</a>  <a href="#">eCompressionFormat = OMX_VIDEO_CodingVP8</a>  <a href="#">eColorFormat = OMX_COLOR_FormatUnused</a>
	<a href="#">OMX_IndexParamVideoPortFormat</a>	r/w	<a href="#">Specify/query the video format.</a> <a href="#">eCompressionFormat = OMX_VIDEO_CodingVP8</a>  <a href="#">eColorFormat = OMX_COLOR_FormatUnused</a>
	<a href="#">OMX_IndexParamVideoVP8</a>	r	<a href="#">eProfile = OMX_VIDEO_VP8ProfileMain</a>  <a href="#">eLevel = OMX_VIDEO_VP8Level_Version0</a>
	<a href="#">OMX_IndexParamVideoProfileLevel_QuerySupported</a>	r	<a href="#">Query supported profile/level pair by index.</a>
	<a href="#">OMX_IndexParamVideoProfileLevel_Current</a>	r	<a href="#">Query current profile/level pair.</a>

<b>Port Index</b>	VPB+1		
<b>Description</b>	Produces uncompressed raw video.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

			<p><u><a href="#">OMX_VIDEO_CodingUnused</a></u></p> <p>eColorFormat = At least one of the following:</p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p>
		r/w	<p><u><a href="#">OMX_IndexParamVideoPortFormat</a></u></p> <p>Specify/query the video format.</p> <p>eCompressionFormat = <u><a href="#">OMX_VIDEO_CodingUnused</a></u></p> <p>eColorFormat = At least one of the following:</p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p>

### 8.8.29.6.2 Video Encoder Class

<b>Name</b>	video_encoder		
<b>Description</b>	Encodes the given uncompressed video stream into a compressed format.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input
	VPB+1	video	output

Upscaling, downscaling and frame rate conversion support is not mandated. If these features are not supported, the component shall implement the slaving behavior as described in section (ref to slaving behavior section).

#### 8.8.2.19.6.2.1 H.263 Encoder Component

<b>Name</b>	video_encoder.h263		
<b>Description</b>	Encodes the given uncompressed video stream into a compressed format.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input
	VPB+1	video	output

<b>Port Index</b>	VPB+0		
<b>Description</b>	Consumes compressed video content.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u>OMX_COLOR_FormatYUV420Planar</u></p> <p><u>OMX_COLOR_FormatYUV420PackedPlanar</u></p> <p><u>OMX_COLOR_FormatYUV420SemiPlanar</u></p> <p><u>OMX_COLOR_FormatYUV420PackedSemiPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420Planar</u></p> <p><u>OMX_COLOR_FormatYVU420PackedPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420SemiPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420PackedSemiPlanar</u></p> <p><u>eColorFormat = OMX_COLOR_FormatYUV420Planar</u></p>

<b>Port Index</b>	VPB+0		
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><b>eColorFormat = At least one of the following:</b></p> <ul style="list-style-type: none"> <li><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></li> </ul> <p>eColorFormat = <u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

<b>Port Index</b>	VPB+1		
<b>Description</b>	Produces compressed video.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>nBitRate = 64000</p> <p>xFrameRate = 15</p> <p>eCompressionFormat = <u><a href="#">OMX_VIDEO_CodingH263</a></u></p> <p>eColorFormat = <u><a href="#">OMX_COLOR_FormatUnused</a></u></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = <u><a href="#">OMX_VIDEO_CodingH263</a></u></p> <p>eColorFormat = <u><a href="#">OMX_COLOR_FormatUnused</a></u></p>

Port Index	VPB+1			
	OMX_IndexParamVideoBitrate	r/w	eControlRate = <i>OMX_Video_ControlRateConstant</i> OMX_Video_ControlRateDisable OMX_Video_ControlRateVariable  nTargetBitrate = 64000	
	OMX_IndexParamVideoErrorCorrection	r/w	bEnableHEC = <i>OMX_TRUE</i>  bEnableResync = <i>OMX_TRUE</i>  nResynchMarkerSpacing = <i>Configureable(0 to 0xFFFFFFFF)</i>	
	OMX_IndexParamVideoH263		eProfile = <i>OMX_VIDEO_H263ProfileBaseline</i>  eLevel= <i>OMX_VIDEO_H263Level10</i>  nPFrames = 0 to 0xffffffff  bPLUSPTYPEAllowed = <i>OMX_FALSE</i>  bForceRoundingTypeToZero = <i>OMX_TRUE</i>  nGOBHeaderInterval = 1 to 9	
	OMX_IndexConfigVideoFramerate	r/w	Specify/query target framerate xFrameRate = 15	
	OMX_IndexConfigVideoBitrate	r/w	Specify/query target bitrate nBitRate = 64000	
	OMX_IndexParamVideoProfileLevel_QuerySupported	r	Query supported profile/level pair by index.	
	OMX_IndexParamVideoProfileLevel_Current	r/w	Specify/query current profile/level pair.	

### 8.8.2.29.6.2.2 AVC Encoder Component

Name	video_encoder.avc			
Description	Encodes the given uncompressed video stream into a compressed format.			
Ports	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	VPB+0	video	input	Consumes the uncompressed raw video content.
	VPB+1	video	output	Produces compressed video.

Port Index	VPB+0		
Description	Consumes compressed video content.		
Required	<b>Index</b>	<b>Access</b>	<b>Description</b>

Port Index	VPB+0		
Parameters/ Configs	Type	Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

<b>Port Index</b>	VPB+1		
<b>Description</b>	Produces compressed video.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>nBitRate = 64000</p> <p>xFrameRate = 15</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingAVC</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingAVC</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoBitrate	r/w	<p>eControlRate =</p> <p><i>OMX_Video_ControlRateConstant</i></p> <p><i>OMX_Video_ControlRateDisable</i></p> <p><i>OMX_Video_ControlRateVariable</i></p> <p>nTargetBitrate = 64000</p>

<b>Port Index</b>	VPB+1		
	OMX_IndexParamVideoAvc	r/w	eProfile = <i>OMX_VIDEO_AVCPProfileBaseline</i> eLevel = <i>OMX_VIDEO_AVCLevel1</i> nSliceHeaderSpacing = <i>Configureable</i> nPFrames = 0 to 0xffffffff bUseHadamard = <i>OMX_TRUE</i> nRefFrames = 1 bEnableFMO = <i>OMX_FALSE</i> bEnableASO = <i>OMX_FALSE</i> bWeightedPPrediction= <i>OMX_FALSE</i> bconstIpred = <i>OMX_FALSE</i>
	OMX_IndexConfigVideoFramerate	r/w	Specify/query target framerate xFrameRate = 15
	OMX_IndexConfigVideoBitrate	r/w	Specify/query target bitrate nBitRate = 64000
	OMX_IndexParamVideoProfileLevel QuerySupported	r	Query supported profile/level pair by index.
	OMX_IndexParamVideoProfileLevel Current	r/w	Specify/query current profile/level pair.

#### **8.8.2.39.6.2.3      MPEG4 Video Encoder Component**

<b>Name</b>	video_encoder.mpeg4		
<b>Description</b>	Encodes the given uncompressed video stream into a compressed format.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input
	VPB+1	video	output

<b>Port Index</b>	VPB+0		
<b>Description</b>	Consumes compressed video content.		
<b>Required</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

Port Index	VPB+0		
Parameters/ Configs	Type	Access	Description
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p> <p><u>eColorFormat =</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p>

<b>Port Index</b>	VPB+1		
<b>Description</b>	Produces compressed video.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the video port settings.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 144</p> <p>nBitRate = 64000</p> <p>xFrameRate = 15</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingMPEG4</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>Specify/query the video format.</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingMPEG4</i></p> <p>eColorFormat = <i>OMX_COLOR_FormatUnused</i></p>
	OMX_IndexParamVideoBitrate	r/w	<p>eControlRate =</p> <p><i>OMX_Video_ControlRateConstant</i></p> <p><i>OMX_Video_ControlRateDisable</i></p> <p><i>OMX_Video_ControlRateVariable</i></p> <p>nTargetBitrate = 64000</p>

<b>Port Index</b>	VPB+1			
	OMX_IndexParamVideoMpeg4	r/w	eProfile = <i>OMX_VIDEO_MPEG4ProfileSimple</i> eLevel = <i>OMX_VIDEO_MPEG4Level1</i> nSliceHeaderSpacing = <i>Configureable</i>  bSVH = <i>OMX_FALSE</i>  bGov = <i>Configureable</i>  nPFrames = 0 to 0xffffffff  nIDCVLCThreshold = 0  bACPred = <i>OMX_TRUE</i>  nHeaderExtension = 1 to 99  bReversibleVLC = <i>OMX_FALSE</i>	
	OMX_IndexConfigVideoFramerate	r/w	Specify/query target framerate xFrameRate = 15	
	OMX_IndexConfigVideoBitrate	r/w	Specify/query target bitrate nBitRate = 64000	
	OMX_IndexParamVideoProfileLevel QuerySupported	r	Query supported profile/level pair by index.	
	OMX_IndexParamVideoProfileLevel Current	r/w	Specify/query current profile/level pair.	

#### **9.6.2.1 VP8 Encoder Component**

<b>Name</b>	video_encoder_vp8			
<b>Description</b>	Encodes the given uncompressed video stream into a compressed format.			
<b>Ports</b>	<b>Index</b> <b>Domain</b> <b>Direction</b> <b>Description</b>			
	VPB+0	video	input	Consumes the uncompressed raw video content.
	VPB+1	video	output	Produces compressed video.

<b>Port Index</b>	VPB+0			
<b>Description</b>	Consumes the uncompressed raw video content.			
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>	
	OMX_IndexParamPortDefinition	r/w	Specify/query the video port settings. <u>nFrameWidth = 176</u>  <u>nFrameHeight = 144</u>  <u>eCompressionFormat = OMX_VIDEO_CodingUnused</u>	

			<p><u>eColorFormat</u> = At least one of the following:</p> <p><u>OMX_COLOR_FormatYUV420Planar</u>  <u>OMX_COLOR_FormatYUV420PackedPlanar</u>  <u>OMX_COLOR_FormatYUV420SemiPlanar</u>  <u>OMX_COLOR_FormatYUV420PackedSemiPlanar</u>  <u>OMX_COLOR_FormatYVU420Planar</u>  <u>OMX_COLOR_FormatYVU420PackedPlanar</u>  <u>OMX_COLOR_FormatYVU420SemiPlanar</u>  <u>OMX_COLOR_FormatYVU420PackedSemiPlanar</u></p>
	<u>OMX_IndexParamVideoPortFormat</u>	r/w	<p><u>Specify/query the video format.</u></p> <p><u>eCompressionFormat</u> = <u>OMX_VIDEO_CodingUnused</u></p> <p><u>eColorFormat</u> = At least one of the following:</p> <p><u>OMX_COLOR_FormatYUV420Planar</u>  <u>OMX_COLOR_FormatYUV420PackedPlanar</u>  <u>OMX_COLOR_FormatYUV420SemiPlanar</u>  <u>OMX_COLOR_FormatYUV420PackedSemiPlanar</u>  <u>OMX_COLOR_FormatYVU420Planar</u>  <u>OMX_COLOR_FormatYVU420PackedPlanar</u>  <u>OMX_COLOR_FormatYVU420SemiPlanar</u>  <u>OMX_COLOR_FormatYVU420PackedSemiPlanar</u></p>

<u>Port Index</u>	<u>VPB+1</u>		
<u>Description</u>	Produces compressed video.		
<u>Required Parameters/Configs</u>	<u>Index</u> <u>OMX_IndexParamPortDefinition</u>	<u>Access</u> r/w	<u>Description</u> <p><u>Specify/query the video port settings.</u></p> <p><u>nFrameWidth = 176</u></p> <p><u>nFrameHeight = 144</u></p> <p><u>nBitRate = 64000</u></p> <p><u>xFrameRate = 15</u></p>

			<u>eCompressionFormat</u> = <u>OMX_VIDEO_CodingVP8</u>
			<u>eColorFormat</u> = <u>OMX_COLOR_FormatUnused</u>
	<u>OMX_IndexParamVideoPortFormat</u>	r/w	Specify/query the video format. <u>eCompressionFormat</u> = <u>OMX_VIDEO_CodingVP8</u>  <u>eColorFormat</u> = <u>OMX_COLOR_FormatUnused</u>
	<u>OMX_IndexParamVideoBitrate</u>	r/w	<u>eControlRate</u> = <u>OMX_Video_ControlRateConstant</u> <u>OMX_Video_ControlRateDisable</u> <u>OMX_Video_ControlRateVariable</u>  <u>nTargetBitrate</u> = 64000
	<u>OMX_IndexParamVideoVP8</u>	r/w	<u>eProfile</u> = <u>OMX_VIDEO_VP8ProfileMain</u>  <u>eLevel</u> = <u>OMX_VIDEO_VP8Level_Version0</u>  <u>nDCTPartitions</u> = 0 to 3  <u>bErrorResilientMode</u> = <u>OMX_FALSE</u> <u>OMX_TRUE</u>
	<u>OMX_IndexConfigVideoFramerate</u>	r/w	Specify/query target framerate <u>xFrameRate</u> = 15
	<u>OMX_IndexConfigVideoBitrate</u>	r/w	Specify/query target bitrate <u>nBitRate</u> = 64000
	<u>OMX_IndexParamVideoProfileLevel_QuerySupported</u>	r	Query supported profile/level pair by index.
	<u>OMX_IndexParamVideoProfileLevel_Current</u>	r/w	Specify/query current profile/level pair.

#### 8.8.39.6.3 Video Reader Class

<b>Name</b>	video_reader		
<b>Description</b>	Reads a video filestream and emits the contained video stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	output

#### 8.8.3.19.6.3.1 Binary Video Reader Component

<b>Name</b>	video_reader.binary		
<b>Description</b>	Blindly reads any video filestream (e.g. a M4V file) irrespective of format and emits contained elementary video stream.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	output

#### **9.6.4 Video Scheduler Class**

Scheduler by definition SHALL support being slave in A/V synchronization. Scheduler may not be required to provide reference clock updates.

<b>Name</b>	video_scheduler			
<b>Description</b>	Times the delivery of video frames according to their timestamps.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	VPB+0	video	input	Accepts video.
	VPB+1	video	output	Emits timed video.
	OPB+0	other/time	input	Accepts time updates.

#### **8.8.3.29.6.4.1 Video Scheduler Component**

<b>Name</b>	video_scheduler.binary			
<b>Description</b>	Times the delivery of video frames according to their timestamps.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	VPB+0	video	input	Accepts video.
	VPB+1	video	output	Emits timed video.
	OPB+0	other/time	input	Accepts time updates.

<b>Port Index</b>	OPB+0			
<b>Description</b>	Accepts media time updates to facilitate accurate emission of a frame at the timestamp for the frame (i.e. in the buffer header). Also provides mechanism for video scheduler to query for media time.			
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>	
	OMX_IndexConfigTimeRenderingDelay	Read /Write	Queries / specifies the new rendering delay	

<b>Port Index</b>	OPB+0			
<b>Description</b>	Accepts media time updates to facilitate accurate emission of a frame at the timestamp for the frame (i.e. in the buffer header). Also provides mechanism for video scheduler to query for media time.			

#### **8.8.49.6.5 Video Writer Class**

<b>Name</b>	video_writer			
<b>Description</b>	Writes given video stream to a video filestream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	VPB+0	video	input	Accepts video stream to be written to the video filestream.

#### **8.8.4.19.6.5.1 Binary Video Writer Class**

<b>Name</b>	video_writer.binary			
-------------	---------------------	--	--	--

<b>Name</b>	video_writer.binary			
<b>Description</b>	Blindly writes given elementary video stream to a video filestream (e.g. an M4V file) irrespective of format.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	VPB+0	video	input	Accepts video stream to be written to the video filestream.

### 8.99.7 Other Standard Components

#### 9.7.1 Camera Class

Video capture components are not mandated to provide reference clock updates.

Although capturing port bits are defined separately for image and video capture they are independent from the bOneShot boolean used to differentiate between image and video modes. By doing so, it becomes possible to capture images in video mode, or to get camera frames of different resolution than preview frames in image mode. If a camera component does not support such advanced features, it returns OMX\_ErrorUnsupportedSetting error to IL client if he tries to trigger a capture in a different camera mode.

In case of video capture - image capture can be triggered concurrently on IPB+0, and in case of still image capture - camera frames of different characteristics than preview/viewfinder frames on VPB+0 can be concurrently available on VPB+1.

<b>Name</b>	camera			
<b>Description</b>	Emits preview/viewfinder video and captured video according to settings.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	VPB+0	video	output	Emits preview/viewfinder video.
	VPB+1	video	output	Emits captured video.
	IPB+0	image	output	Emits captured images.
	OPB+0	other/time	input	Receives media time update/provides access to clock component.

#### 8.9.1.19.7.1.1 YUV Camera Component

<b>Name</b>	camera.yuv			
<b>Description</b>	Emits preview/viewfinder video and captured video according to settings.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	VPB+0	video	output	Emits preview/viewfinder video.
	VPB+1	video	output	Emits captured video.
	IPB+0	image	output	Emits captures images.
	OPB+0	other/time	input	Receives media time update/provides access to clock component.

<b>Port Index</b>	OMX_ALL
-------------------	---------

<b>Port Index</b>	OMX_ALL		
<b>Description</b>	Properties that apply to all ports.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamCommonSensorMode	r/w	Specifies the sensor mode. <i>The bOneShot field indicates whether the camera will emit a single frame or a stream of frames when capturing. The camera resolution should not be changed by the IL client be left as the default value. So the camera may change the sensor set the resolution as needed to satisfy the resolution specified on the according to resolution of output ports.</i>
	OMX_IndexConfigCommonWhiteBalance	r/w	Specifies white balance
	OMX_IndexConfigCommonDigitalZoom	r/w	Specifies digital zoom
	OMX_IndexConfigCommonExposureValue	r/w	Specifies exposure value compensation
	<i>OMX_IndexAutoPauseAfterCapture</i> <i>OMX_IndexConfigCapturing</i>	<i>r/w/rw</i>	<i>Specifies whether the camera will automatically transition to OMX_StatePause after the Capturing boolean is cleared (e.g. to facilitate a frozen viewfinder). Specifies whether the camera is emitting captured data or not.</i>
	<i>OMX_IndexAutoPauseAfterCapture</i>	<i>#w</i>	<i>Specifies whether the camera will automatically transition to OMX_StatePaused after the Capturing boolean is cleared (e.g. to facilitate a frozen viewfinder).</i>

<b>Port Index</b>	VPB+0		
<b>Description</b>	Emits preview/viewfinder video when the camera component is executing.		
<b>Required</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

Port Index	VPB+0		
Parameters/ Configs	OMX_IndexParamPortDefinition	r/w	
	<p>Specifies preview's resolution <del>and framerate</del>.</p> <p>nFrameWidth = 320</p> <p>nFrameHeight = 240</p> <p><del>nStride = 320</del></p> <p><del>nSliceHeight = 16</del></p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p>eColorFormat = At least one of the following:</p> <ul style="list-style-type: none"> <li><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></li> </ul> <p>eColorFormat = <del>OMX_COLOR_FormatYUV420Planar</del></p>		

<b>Port Index</b>	VPB+0		
	OMX_IndexParamVideoPortFormat	r/w	<p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u>OMX_COLOR_FormatYUV420Planar</u></p> <p><u>OMX_COLOR_FormatYUV420PackedPlanar</u></p> <p><u>OMX_COLOR_FormatYUV420SemiPlanar</u></p> <p><u>OMX_COLOR_FormatYUV420PackedSemiPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420Planar</u></p> <p><u>OMX_COLOR_FormatYVU420PackedPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420SemiPlanar</u></p> <p><u>OMX_COLOR_FormatYVU420PackedSemiPlanar</u></p> <p>eColorFormat = <u>OMX_COLOR_FormatYUV420Planar</u></p>

<b>Port Index</b>	VPB+1
<b>Description</b>	Emits captured video <del>when the camera component is capturing where the number of output frames depends on the sensor mode. If the sensor mode is set to one shot then this port only emits a one frame per capture. Output may be interpreted as raw image.</del>
<b>Formats</b>	OMX_VIDEO_CodingUnused
<b>Required</b>	Index
	Access    Description

<b>Port Index</b>	VPB+1		
<b>Parameters/ Configs</b>	OMX_IndexParamPortDefinition	r/w	<p>Specifies emitted video's resolution <del>and framerate</del>.</p> <p>nFrameWidth = 640</p> <p>nFrameHeight = 480</p> <p><del>nStride = 640</del></p> <p><del>nSliceHeight = 16</del></p> <p>eCompressionFormat = OMX_VIDEO_CodingUnused</p> <p>eColorFormat = At least one of the following:</p> <ul style="list-style-type: none"> <li><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></li> <li><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></li> </ul> <p>eColorFormat = <del>OMX_COLOR_FormatYUV420Planar</del></p>

<b>Port Index</b>	VPB+1		
	<u><a href="#">OMX_IndexParamVideoPortFormat</a></u> <u><a href="#">OMX_IndexConfigCommonPortCapturing</a></u>	<u><a href="#">r/w</a></u> <u><a href="#">r/w</a></u>	<p><u>eCompressionFormat = OMX_VIDEO_CodingUnused</u></p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u></p> <p><u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p>
			<p><u>Capturing port bit that controls video capture.</u></p> <p><u><a href="#">OMX_CONFIG_PORTBOOLEANTYPE</a></u></p> <p><u>Port specific capture boolean.</u></p>

<b>Port Index</b>	IPB+0		
<b>Description</b>	Emits captured images.		
<b>Required</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

<b>Port Index</b>	<b>IPB+0</b>		
<b>Parameters/ Configs</b>	<a href="#">OMX_IndexParamPortDefinition</a>	r/w	<p><u>Specifies image's resolution.</u>  <u>nFrameWidth = 640</u>  <u>nFrameHeight = 480</u></p> <p><u>eCompressionFormat =</u>  <u>OMX IMAGE_CodingUnused</u></p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p>
	<a href="#">OMX_IndexParamImagePortFormat</a>	r/w	<p><u>eCompressionFormat =</u>  <u>OMX IMAGE_CodingUnused</u></p> <p><u>eColorFormat = At least one of the following:</u></p> <p><u><a href="#">OMX_COLOR_FormatYUV420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYUV420PackedSemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420Planar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420SemiPlanar</a></u>  <u><a href="#">OMX_COLOR_FormatYVU420PackedSemiPlanar</a></u></p>
	<a href="#">OMX_IndexConfigCommonPortCapture</a>	r/w	<p><u>Capturing port bit that controls image capture.</u>  <u>OMX_CONFIG_PORTBOOLEANTYPE</u>  <u>Port specific capture boolean.</u></p>

<b>Port Index</b>	OPB+0
<b>Description</b>	Accepts media time updates. Provides mechanism for camera component to query for media time. Camera component can detect drift between camera clock and media clock (which may use the audio capturer as a master) and correct timestamps on outgoing frames to compensate. In use case where two cameras are used (e.g. one pointed at user and one pointed away) this provides a consistent media time for timestamps across switches between cameras during capture.

### **8.9.1.29.7.1.2 Video Capture Use Case**

An IL client using a camera to capture a video stream may do so via the following steps:

1. Instantiate the camera component and any co-operating components.
2. Set camera parameters:
  - a. Set capture port resolution and frame-rate according to desired values of captured stream
  - b. Set viewfinder port resolution and frame-rate (e.g. according to desired values of preview window)
  - c. Clear the one shot bit of the sensor mode to indicate that the camera should emit a stream of multiple frames, i.e. a video stream. The IL client should leave the sensor resolution at the default allowing the camera to pick a sensor resolution appropriate to the resolution settings of the viewfinder and capture ports.
  - d. Set other camera settings (e.g. exposure value compensation, white balance, zoom, etc).
  - e. Set or clear auto pause after capture accordingly. If auto pause is set the component will pause and the viewfinder will freeze after a capture.
3. Establish any necessary tunnels between the camera component and other components (e.g. a display component tunneling with the viewfinder port or a video encoder tunneling with the capture port).
4. Select the clock component's active reference clock. If the camera is used in concert with an audio capturer the audio clock will be the active reference clock (i.e. be the master clock) to facilitate synchronized audio/video capture. Otherwise the video clock provided by the camera will be the active reference clock.
5. Transition all components to the OMX\_StateIdle state and then to the OMX\_StateExecuting state. The viewfinder port should now be actively emitting preview frames.
6. To initiate video capture set the capturing bit. The capture port will emit captured frames at the frame rate specified. If using a clock component start the clock component. Timestamps applied to video frames will follow the media

- time to facilitate consistent timestamp authoring between audio and video capture. The viewfinder will continue to emit frames.
7. To terminate video capture clear the capturing bit. The capture port will cease the emission of frames. If set to auto pause the component will pause and the viewfinder will cease the emission of frames. This effectively freezes any associated preview window to the last frame emitted which should be identical to the last frame emitted by the capture port. If auto pause is clear then the viewfinder continues emitting preview frames.
  8. If the component is paused and the viewfinder is frozen after a capture then the IL client manually unfreezes the viewfinder by transitioning the component to OMX\_StateExecuting when appropriate (e.g. after the captured video has been stored by the application).

Note that this sequence of calls can also be used to implement a sequence of consecutive image captures. In the case of a sequence of stills the IL client simply sets the frame rate on the capture port to accommodate the desired interim between captured stills, uses a JPEG encoder instead of an MPEG encoder, and terminates the capture after the desired number of stills have been captured.

#### **8.9.1.39.7.1.3 Still Image Capture**

An IL client using a camera to capture an image may do so via the following steps:

1. Instantiate the camera component and any co-operating components
2. Set camera parameters:
  - a. Set capture port resolution according to desired values of captured image.
  - b. Set viewfinder port resolution and frame-rate (e.g. according to desired values of preview window).
  - c. Set the one shot bit of the sensor mode to indicate that the camera should emit a single frame, i.e. an image frame. The IL client should leave the sensor resolution at the default allowing the camera to pick a sensor resolution appropriate to the resolution settings of the viewfinder and capture ports.
  - d. Set other camera settings (e.g. exposure value compensation, white balance, zoom, etc).
  - e. Set or clear auto pause after capture accordingly. If auto pause is set the component will pause and the viewfinder will freeze after a capture.
3. Establish any necessary tunnels between the camera component and other components (e.g. a display component tunneling with the viewfinder port or an image encoder tunneling with the capture port).
4. Transition all components to the OMX\_StateIdle state and then to the OMX\_StateExecuting state. The viewfinder port should now be actively

- emitting preview frames and the capture port is not transmitting any frames, it is paused.
5. With the viewfinder port enabled, the IL client now has the opportunity to perform any zoom and focus related actions.
  6. To signal image capture set the capturing bit. The capture port will emit a single captured frame and then the component will immediately clear the capturing bit. If set to auto pause after capture the component will transition itself to the OMX\_StatePause state and the viewfinder will cease the emission of frames. This effectively freezes any associated preview window to the captured image frame. If auto pause is clear then the viewfinder continues emitting preview frames.
  7. If the component is paused and the viewfinder is frozen after a capture then the IL client manually unfreezes the viewfinder by transitioning the component to OMX\_StateExecuting when appropriate (e.g. after the captured image has been stored by the application).

### 8.9.29.7.2 Clock Class

<b>Name</b>	Clock			
<b>Description</b>	Implements the OpenMAX IL clock component (add reference to existing section in spec describing the clock component), the component may expose support for 1 to N ports.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	OPB+0 to (OPB+N-1)	other/time	output	Emits time updates.

### 8.9.2.19.7.2.1 Clock Component

<b>Name</b>	clock.binary
<b>Description</b>	Implements the OpenMAX IL clock component.

<b>Port Index</b>	OPB+0 to (OPB+N-1)		
<b>Description</b>	Emits time updates.		
<b>Formats</b>	OMX_OTHER_FormatTime		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexConfigTimeScale	Read, write	Query or set current scale applied to the media time.
	OMX_IndexConfigTimeClockState	Read, write	Query or set current clock state.
	OMX_IndexConfigTimeActiveRefClock	Read, write	Query or set the active reference clock.
	OMX_IndexConfigTimeCurrentMediaTime	Read	Query current media time.

<b>Port Index</b>	OPB+0 to (OPB+N-1)		
	OMX_IndexConfigTimeCurrentWallTime	Read	Query current wall clock time.
	OMX_IndexConfigTimeCurrentAudioReference	Write	Set the instantaneous <del>audio</del> reference clock value.
	OMX_IndexConfigTimeMediaTimeRequest	Write	Make a media time request.
	OMX_IndexConfigTimeClientStartTime	Write	Set the start time of a client stream.
	<del>OMX_IndexConfigTimeRenderingDelay</del>	<u>Write</u>	<u>Set the new rendering delay for the clock client</u>

### 8.9.39.7.3 Container Demuxer Class

<b>Name</b>	container_demuxer		
<b>Description</b>	Parses a container filestream, demuxes its elementary streams, and emits them as independent video, image and audio streams.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	APB+0	audio	output
	VPB+0	video	output
	OPB+0	other/time	input
Emits demuxed audio stream.			
Emits demuxed video stream.			
Receives media time <del>updates</del> /provides access to clock component.			

<b>Port Index</b>	OMX_ALL		
<b>Description</b>	Properties that apply to all ports.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexConfigTimePosition	r/w	Specifies the position in the container format content.
	OMX_IndexConfigTimeSeekMode	r/w	Specifies the manner in which a seek will be carried out (quickly or precisely).
	OMX_IndexParamContentURI	r/w	Specify/query the current target content.

<b>Port Index</b>	APB+0		
<b>Description</b>	Emits demuxed audio stream.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the characteristics of the audio stream.
	OMX_IndexParamNumAvailableStreams	r	Query the number of available audio streams for this port given current content.

<b>Port Index</b>	APB+0		
	OMX_IndexParamActiveStream	r/w	Specify/query the active audio stream by index where indices are numbered from 0 to the number of available streams.

<b>Port Index</b>	VPB+0		
<b>Description</b>	Emits demuxed video stream.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the characteristics of the video stream.
	OMX_IndexParamNumAvailableStreams	r	Query the number of available video streams for this port given current content.
	OMX_IndexParamActiveStream	r/w	Specify/query the active video stream by index where indices are numbered from 0 to the number of available streams.

<b>Port Index</b>	OPB+0		
<b>Description</b>	Accepts media time updates. Provides mechanism for component to query for media time. The demuxer obeys changes in the media time to implement trick modes. For instance a negative media time scale factor indicates rewind which implies the demuxer shall retrieve data in reverse order.		

### 8.9.3.19.7.3.1 Playback Use Case

An IL client using a container parser to playback content may do so via the following steps:

1. Instantiate the container demuxer component.
2. Set any relevant container demuxer settings:
3. Specify the target content
4. set all outputs to autodetect
5. Execute the component until all each port generates an OMX\_EventPortSettingsChanged event. For each port that generates this event:
  - a. Query the number of available streams for that port and examine the properties of each available stream by making each active and reading the port parameters.
  - b. Make the desired stream active.

6. Instantiate the set of co-operating components appropriate to the format settings of the parser's output ports.
7. Establish any necessary tunnels between the container parser and component and other components (e.g. an audio decoder tunneling with the audio port or a video decoder tunneling with the video port).
8. Select the clock component's active reference clock. In a use case with audio this is normally the audio clock as provided by the audio renderer.
9. Transition all components to the OMX\_StateIdle state then the OMX\_StateExecuting state. If using a clock component start the clock component. The container demuxer will emit the relevant elementary streams facilitating playback.
10. To change the playback rate (i.e. facilitate trick modes) change the media clock scale factor to the appropriate value (e.g. 2.0 implies 2x forward playback and -1.0 implies 1x reverse playback). The clock component will inform the container demuxer of the scale change and the demuxer will retrieve and emit data in a manner appropriate the scale (e.g. in reverse for negative scales or skipping interframes in extreme fast forward).
11. To seek to a particular location the IL client sets the position on the container demux.

#### **8.9.3.29.7.3.2 3GP Demuxer Component**

The standard 3GP demuxer component shall support Release 6 of the 3GP format including basic profile (all other profiles are optional).

#### **8.9.3.39.7.3.3 ASF Demuxer Component**

The standard ASF demuxer component shall support ASF version 1.2, Revision 1.20.03 (dated December 2004)

#### **8.9.3.49.7.3.4 Real Demuxer Component**

The standard Real Demuxer shall support parsing of the Real container format.

#### **8.9.49.7.4 Container Muxer Class**

<b>Name</b>	container_muxer			
<b>Description</b>	Given independent video, image, and audio streams muxes them into a container filestream.			
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>	<b>Description</b>
	APB+0	audio	input	Accepts audio stream for muxing.
	VPB+0	video	input	Accepts video stream for muxing.

<b>Port Index</b>	OMX_ALL
-------------------	---------

<b>Port Index</b>	OMX_ALL		
<b>Description</b>	Properties that apply to all ports.		
<b>Required Parameters/Configs</b>	<b>Index</b> OMX_IndexParamContentURI	<b>Access</b> r/w	<b>Description</b> Specify/query the current target content.

#### **8.9.4.19.7.4.1 3GP Muxer Component**

The standard 3GP muxer component shall support Release 6 of the 3GP format including basic profile (all other profiles are optional).

#### **8.9.59.7.5 Image/Video Processor Class**

<b>Name</b>	iv_processor		
<b>Description</b>	Performs some processing on a raw image/video stream.		
<b>Ports</b>	<b>Index</b> VPB+0	<b>Domain</b> video	<b>Direction</b> input Accepts video for processing.
	VPB+1	video	<b>Direction</b> output Emits processed video.

Upscaling, downscaling and frame rate conversion support is not mandated. If these features are not supported, the component shall implement the slaving behavior as described in section (ref to slaving behavior section).

#### **8.9.5.19.7.5.1 YUV Image/Video Processor**

<b>Name</b>	iv_processor.yuv		
<b>Description</b>	Performs some processing on a raw image/video stream.		
<b>Ports</b>	<b>Index</b> VPB+0	<b>Domain</b> video	<b>Direction</b> input Accepts video for processing.
	VPB+1	video	<b>Direction</b> output Emits processed video.

<b>Port Index</b>	VPB+0		
<b>Description</b>	Accepts video for processing.		
<b>Required</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

Port Index	VPB+0		
<b>Parameters/ Configs</b>	OMX_IndexParamPortDefinition	r/w	<p>Specify/query the characteristics of the video stream.</p> <p>nFrameWidth = 640</p> <p>nFrameHeight = 480</p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i></p> <p><b>eColorFormat = At least one of the following:</b></p> <p><i>OMX_COLOR_FormatYUV420Planar</i>  <i>OMX_COLOR_FormatYUV420PackedPlanar</i>  <i>OMX_COLOR_FormatYUV420SemiPlanar</i>  <i>OMX_COLOR_FormatYVU420Planar</i>  <i>OMX_COLOR_FormatYVU420PackedPlanar</i>  <i>OMX_COLOR_FormatYVU420SemiPlanar</i>  <i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i></p> <p><b>eColorFormat =</b>  <b><i>OMX_COLOR_FormatYUV420Planar</i></b></p>
	OMX_IndexParamVideoPortFormat	r/w	<p>eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i></p> <p><b>eColorFormat = At least one of the following:</b></p> <p><i>OMX_COLOR_FormatYUV420Planar</i>  <i>OMX_COLOR_FormatYUV420PackedPlanar</i>  <i>OMX_COLOR_FormatYUV420SemiPlanar</i>  <i>OMX_COLOR_FormatYVU420Planar</i>  <i>OMX_COLOR_FormatYVU420PackedPlanar</i>  <i>OMX_COLOR_FormatYVU420SemiPlanar</i>  <i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i></p> <p><b>eColorFormat =</b>  <b><i>OMX_COLOR_FormatYUV420Planar</i></b></p>

<b>Port Index</b>	VPB+1		
<b>Description</b>	Emits processed video.		
<b>Required Parameters/Configs</b>	<b>Index</b> OMX_IndexParamPortDefinition	<b>Access</b> r/w	<b>Description</b> Specify/query the characteristics of the video stream. nFrameWidth = 640 nFrameHeight = 480 (output width and height imply scale if different then input width and height) eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i>  <u>eColorFormat = At least one of the following:</u> <u><i>OMX_COLOR_FormatYUV420Planar</i></u> <u><i>OMX_COLOR_FormatYUV420PackedPlanar</i></u> <u><i>OMX_COLOR_FormatYUV420SemiPlanar</i></u> <u><i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i></u> <u><i>OMX_COLOR_FormatYVU420Planar</i></u> <u><i>OMX_COLOR_FormatYVU420PackedPlanar</i></u> <u><i>OMX_COLOR_FormatYVU420SemiPlanar</i></u> <u><i>OMX_COLOR_FormatYVU420PackedSemiPlanar</i></u> <u>eColorFormat =</u> <u><i>OMX_COLOR_FormatYUV420Planar</i></u>

Port Index	VPB+1		
	OMX_IndexParamVideoPortFormat	r/w	<p>eCompressionFormat =  <i>OMX_VIDEO_CodingUnused</i></p> <p>eColorFormat = At least one of the following:  <i>OMX_COLOR_FormatYUV420Planar</i>  <i>OMX_COLOR_FormatYUV420PackedPlanar</i>  <i>OMX_COLOR_FormatYUV420SemiPlanar</i>  <i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i>  <i>OMX_COLOR_FormatYVU420Planar</i>  <i>OMX_COLOR_FormatYVU420PackedPlanar</i>  <i>OMX_COLOR_FormatYVU420SemiPlanar</i>  <i>OMX_COLOR_FormatYVU420PackedSemiPlanar</i>  eColorFormat =  <i>OMX_COLOR_FormatYUV420Planar</i></p>
	OMX_IndexConfigCommonRotate	r/w	<p>Specify/query rotation. Rotation is always performed prior to mirror.</p> <p>nRotation =  0  90 (-270)  180 (-180)  270 (-90)</p>
	OMX_IndexConfigCommonMirror	r/w	<p>eMirror =  <i>OMX_MirrorNone</i>  <i>OMX_MirrorVertical</i>  <i>OMX_MirrorHorizontal</i>  <i>OMX_MirrorBoth</i></p>
	OMX_IndexConfigCommonInputCrop	r/w	<p>Cropping shall be specified within frame boundaries:</p> <p>0&lt;= nLeft &lt;= frame width -1  0&lt;= nTop &lt;= frame height -1  0&lt;= nWidth &lt;= frame width  0&lt;= nHeight &lt;= frame height  Cropping is 16-byte aligned.</p>

#### 8.9.69.7.6 Image/Video Renderer Class

Name	iv_renderer			
Description	Displays a given raw image/video stream.			
Ports	Index	Domain	Direction	Description
	VPB+0	video	input	Accepts video for display.

Common to all renderers:

<b>Port Index</b>	VPB+0		
<b>Description</b>	Accepts video rendering.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexConfigCommonRotate	r/w	Specify/query rotation. Rotation is always performed prior to mirror. nRotation = 0 90 (-270) 180 (-180) 270 (-90)
	OMX_IndexConfigCommonMirror	r/w	eMirror = <i>OMX_MirrorNone</i> <i>OMX_MirrorVertical</i> <i>OMX_MirrorHorizontal</i> <i>OMX_MirrorBoth</i>
	OMX_IndexConfigCommonScale	r/w	xWidth = downscale factors of 2 xHeight = downscale factors of 2
	OMX_IndexConfigCommonInputCrop	r/w	Cropping shall be specified within frame boundaries: 0<= nLeft <= frame width -1 0<= nTop <= frame height -1 0<= nWidth <= frame width 0<= nHeight <= frame height Cropping is 16-byte aligned.
	<a href="#">OMX_IndexConfigTimeRenderingDelay</a>	r	Queries the rendering delay

#### 8.9.6.19.7.6.1 YUV Overlay Image/Video Renderer

<b>Name</b>	iv_renderer.yuv.overlay		
<b>Description</b>	Displays a given raw yuv image/video stream using overlays.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input

<b>Port Index</b>	VPB+0		
<b>Description</b>	Accepts video rendering.		
<b>Required</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>

Port Index	VPB+0		
Parameters/ Configs	OMX_IndexParamPortDefinition	r/w	
	<p>Specify/query the characteristics of the video stream.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 220</p> <p><del>nStride = 16</del></p> <p><del>nSliceHeight = 16</del></p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i></p> <p>eColorFormat = <u>At least one of the following:</u></p> <p><u><i>OMX_COLOR_FormatYUV420Planar</i></u></p> <p><u><i>OMX_COLOR_FormatYUV420PackedPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYUV420SemiPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYVU420Planar</i></u></p> <p><u><i>OMX_COLOR_FormatYVU420PackedPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYVU420SemiPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYVU420PackedSemiPlanar</i></u></p> <p>eColorFormat = <del><i>OMX_COLOR_FormatYUV420Planar</i></del></p>		

Port Index	VPB+0		
	OMX_IndexParamVideoPortFormat	r/w	<p>eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i></p> <p>eColorFormat = At least one of the following:</p> <p><i>OMX_COLOR_FormatYUV420Planar</i>  <i>OMX_COLOR_FormatYUV420PackedPlanar</i>  <i>OMX_COLOR_FormatYUV420SemiPlanar</i>  <i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i>  <i>OMX_COLOR_FormatYVU420Planar</i>  <i>OMX_COLOR_FormatYVU420PackedPlanar</i>  <i>OMX_COLOR_FormatYVU420SemiPlanar</i>  <i>OMX_COLOR_FormatYVU420PackedSemiPlanar</i>  eColorFormat = <i>OMX_COLOR_FormatYUV420Planar</i></p>

#### 8.9.6.29.7.6.2 YUV BLTter Image/Video Renderer

Name	iv_renderer.yuv.blter		
Description	Displays a given raw yuv image/video stream via bitBLTs.		
Ports	Index	Domain	Direction
	VPB+0	video	input

Port Index	VPB+0		
Description	Accepts video rendering.		
Required	Index	Access	Description

Port Index	VPB+0		
Parameters/ Configs	OMX_IndexParamPortDefinition	r/w	
			<p>Specify/query the characteristics of the video stream.</p> <p>nFrameWidth = 176</p> <p>nFrameHeight = 220</p> <p><del>nStride = 16</del></p> <p><del>nSliceHeight = 16</del></p> <p>eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i></p> <p>eColorFormat = <u>At least one of the following:</u></p> <p><u><i>OMX_COLOR_FormatYUV420Planar</i></u></p> <p><u><i>OMX_COLOR_FormatYUV420PackedPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYUV420SemiPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYVU420Planar</i></u></p> <p><u><i>OMX_COLOR_FormatYVU420PackedPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYVU420SemiPlanar</i></u></p> <p><u><i>OMX_COLOR_FormatYVU420PackedSemiPlanar</i></u></p> <p>eColorFormat = <del><i>OMX_COLOR_FormatYUV420Planar</i></del></p>

Port Index	VPB+0			
	OMX_IndexParamVideoPortFormat	r/w	eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i>  eColorFormat = At least one of the following: <i>OMX_COLOR_FormatYUV420Planar</i> <i>OMX_COLOR_FormatYUV420PackedPlanar</i> <i>OMX_COLOR_FormatYUV420SemiPlanar</i> <i>OMX_COLOR_FormatYUV420PackedSemiPlanar</i> <i>OMX_COLOR_FormatYVU420Planar</i> <i>OMX_COLOR_FormatYVU420PackedPlanar</i> <i>OMX_COLOR_FormatYVU420SemiPlanar</i> <i>OMX_COLOR_FormatYVU420PackedSemiPlanar</i> eColorFormat = <i>OMX_COLOR_FormatYUV420Planar</i>	

#### 8.9.6.39.7.6.3 RGB Overlay Image/Video Renderer

Name	iv_renderer.rgb.overlay		
Description	Displays a given raw rgb image/video stream using overlays.		
Ports	Index	Domain	Direction
VPB+0	video	input	Accepts video for display.

Port Index	VPB+0			
Description	Accepts video rendering.			
Required Parameters/Configs	Index	Access	Description	
	OMX_IndexParamPortDefinition	r/w	eSpecify/query the characteristics of the video stream. nFrameWidth = 176  nFrameHeight = 220  <i>nStride = 352 (176 pixels @ 16 bpp)</i>  <i>nSliceHeight = 16</i>  eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i>  eColorFormat = <i>OMX_COLOR_Format16bitRGB565</i>	

<b>Port Index</b>	VPB+0		
	OMX_IndexParamVideoPortFormat	r/w	eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i>  eColorFormat = <i>OMX_COLOR_Format16bitRGB565</i>

#### 8.9.6.49.7.6.4 RGB BLTter Image/Video Renderer

<b>Name</b>	iv_renderer.rgb.blter		
<b>Description</b>	Displays a given raw rgb image/video stream vis bitBLTS.		
<b>Ports</b>	<b>Index</b>	<b>Domain</b>	<b>Direction</b>
	VPB+0	video	input Accepts video for display.

<b>Port Index</b>	VPB+0		
<b>Description</b>	Accepts video rendering.		
<b>Required Parameters/Configs</b>	<b>Index</b>	<b>Access</b>	<b>Description</b>
	OMX_IndexParamPortDefinition	r/w	Specify/query the characteristics of the video stream. nFrameWidth = 176  nFrameHeight = 220  <del>nStride = 352 (176 pixels @ 16 bpp)</del>  <del>nSliceHeight = 16</del>  eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i>  eColorFormat = <i>OMX_COLOR_Format16bitRGB565</i>
	OMX_IndexParamVideoPortFormat	r/w	eCompressionFormat = <i>OMX_VIDEO_CodingUnused</i>  eColorFormat = <i>OMX_COLOR_Format16bitRGB565</i>

## **9 Content Pipes**

### **9.1 Rationale**

Streaming media processing requires efficient data flow in and out of a media processing object.

For instance, in the playback use case a container format parser/demuxer typically pulls source data in a manner that assumes reads on a local file. Likewise, in the recording use case, a container format combiner/muxer typically pushes final data in a manner that assumes writes on a local file. Such “file access” is usually synchronous and includes some high frequency reads/writes of small size as well as random access.

In some cases, the content from which source data is pulled from or which final data is pushed to is not local or is not from a file. This conventional approach to this use case, often referred to as “data” streaming, leverages queues of large input or output buffers of linear data transferred asynchronously. This model is at odds with the model parsers and combiners expect. If conventional streaming is used then reconciling the two transfer models involves inefficient (and unnecessary) memory copies, waiting, and complexity.

### **9.2 Concept**

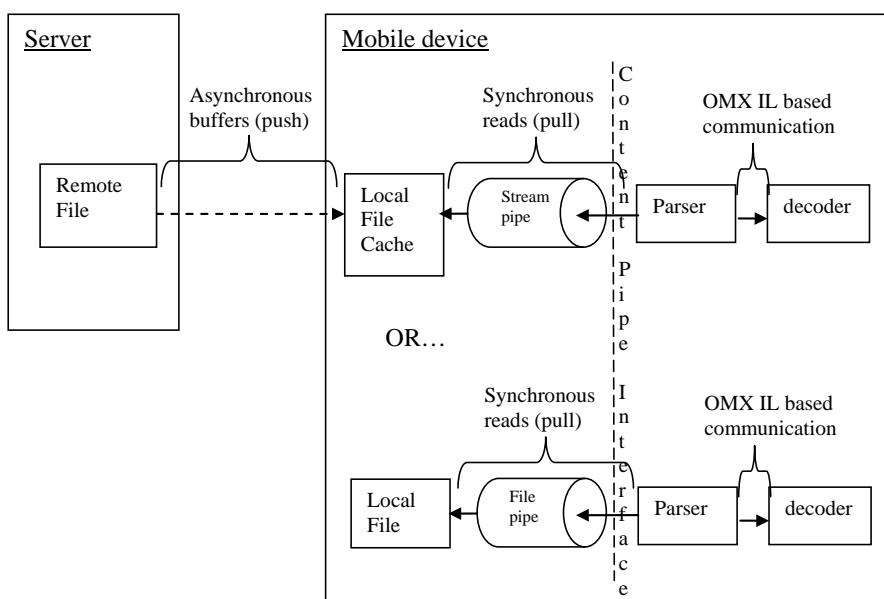
We eliminate the inconsistency of these models by constructing a data access abstraction interface for pulling source data and pushing final data that lends itself to the needs of parsers and combiners. Rather than restricting ourselves to “file access” and the connotations it implies we use a more generalized notion of “content piping”.

A “content pipe” is an abstraction for any mechanism of accessing content data (i.e. pulling content data in or pushing content data out). This abstraction is not tied any particular implementation. A pipe may be implemented, for example, as a local file, a remote file, a broadcast stream, memory buffers, intermediate data from derived from persistent data, etc. A pipe needn’t be limited to a single method of providing access. For instance a single pipe may provide via both local files and remote files, or through multiple transport protocols. A system may include one or many pipes.

### **9.3 Implementation**

Since content pipe functions are synchronous, the implementation of the pipe interface is local even if the content itself is remote. This may entail a local agent acting as a broker between asynchronously pushed buffers from remote content and a pipe client (e.g. a parser) that must synchronously pull in data of varying sizes. Such an agent would maintain both the complex/elastic connection between the remote content and a local cache (which entails careful synchronization) as well as the simple/rigid connection between the local cache and the parser (which as a pull interface lacks complex synchronization).

Note that the synchronous pull based transfer implied by content pipe interface implies neither that the physical connection to the content nor the propagation of the data beyond the client be synchronous and pull based. For example consider the example of an OpenMAX IL parser component reading from either a remote file or a local one. The parser is provided the interface it requires, the mechanism to satisfy the pipe is completely abstracted and may actually use asynchronous data transfers, and the downstream data transfer is completely unaffected.



**Figure 9-1. Content Pipe Operation**

## 9.4 Definition

### 9.4.1 Content Access and Manipulation

The pipe interface includes functions for opening, creating and closing content handles:

```

CResult (*Open)( CPhandle *hContent, CPstring szURI, CP_ACCESSTYPE eAccess );
CResult (*Close)( CPhandle hContent );
CResult (*Create)( CPhandle *hContent, CPstring szURI );

```

Because content parsers and muxers operate as though they are accessing files directly, a pipe's data access functions are modeled on conventional file access. These include

~~functions for reading and writing data using client buffers and setting/retrieving the read/write position within the content:~~

```
CPResult (*SetPosition)( CPhandle hContent, CPint nOffset, CP_ORIGINTYPE eOrigin);  
CPResult (*GetPosition)( CPhandle hContent, *pPosition);  
CPResult (*Read)( CPhandle hContent, CPbyte *pData, CPuint nSize);  
CPResult (*Write)( CPhandle hContent, CPbyte *pData, CPuint nSize);
```

#### **9.4.2 Streaming Support**

This proposal recognizes that the source content may be remotely located and streamed during processing to a position of local accessibility (e.g. a local cache of remote content). The pipe interface includes a set of functions to accommodate such scenarios.

The `CheckAvailableBytes` function queries if a given number of bytes are available. This allows the client to check for the availability of enough bytes to satisfy a large section of parsing prior to beginning the parsing. This allows a pipe implementation to stream data to a local cache.

```
CPResult (*CheckAvailableBytes)( CPhandle hContent, CPuint nBytesRequested,  
CP_CHECKBYTESRESULTTYPE *eResult);
```

If the bytes are not immediately available the pipe will call the client via the provided callback when they are. This callback mechanism also includes events for data overflow and a pipe disconnection (e.g. if the connection with a remote source is lost). See the `CP_EVENTTYPE` enumeration for details.

The `ReadBuffer` function reads a large area of data using the pipe implementation's memory. If a pipe implementation is streaming remote data to a local cache the desired data will already reside in local memory prior to a call on this function. This function avoids the memory copy that would be required if the client provided the memory pointer. Instead, this function allows the pipe implementation to provide the memory pointer.

```
CPResult (*ReadBuffer)( CPhandle hContent, CPbyte **ppBuffer, CPuint *nSize,  
CPbool bForbidCopy);
```

This necessitates a `ReleaseReadBuffer` function to release a buffer acquired via `ReadBuffer`.

```
CPResult (*ReleaseReadBuffer)( CPhandle hContent, CPbyte *pBuffer);
```

The `WriteBuffer` function to writes a large area of data using the pipe implementation's memory without imposing an unnecessary copy.

```
CPResult (*GetWriteBuffer)( CPhandle hContent, CPbyte **ppBuffer, CPuint  
nSize);
```

This necessitates the `GetWriteBuffer` function to acquire a write buffer for use with `WriteBuffer`.

```
CPResult (*WriteBuffer)( CPhandle hContent, CPbyte *pBuffer, CPuint  
nFilledSize);
```

### **9.4.3 *Enumerations***

#### **9.4.3.1 CP\_ORIGINTYPE**

The **CP\_ORIGINTYPE** enumeration defines all the origin types used by the **SetPosition** method of the **CP\_PIPE** from which the indicated position is relative.

**Table 10-9-1: Content Pipe Origin Types**

<b>Value</b>	<b>Description</b>
<b>CP_OriginBegin</b>	Origin is the beginning of content, specifically the first byte of the content's data stream.
<b>CP_OriginCur</b>	Origin is the current position within the content.
<b>CP_OriginEnd</b>	Origin is the beginning of content, specifically the last byte of the content's data stream.

#### **9.4.3.2 CP\_ACCESTYPE**

The **CP\_ACCESTYPE** enumeration defines all the access types used by the **Open** method of the **CP\_PIPE**.

**Table 10-9-2: Content Pipe Access Types**

<b>Value</b>	<b>Description</b>
<b>CP_AccessRead</b>	Access type is read only.
<b>CP_AccessWrite</b>	Access type is write only.
<b>CP_AccessReadWrite</b>	Access type is both read and write.

#### **9.4.3.3 CP\_CHECKBYTESRESULTTYPE**

The **CP\_CHECKBYTESRESULTTYPE** enumeration defines all possible results of a call to the **CheckAvailableBytes** method of the **CP\_PIPE**.

**Table 10-9-3: Content Pipe CheckAvailableBytes Result Types**

<b>Value</b>	<b>Description</b>
<b>CP_CheckBytesOk</b>	There are at least the requested number of bytes available.
<b>CP_CheckBytesNotReady</b>	The pipe is still retrieving bytes and presently lacks sufficient bytes. Client will be called when sufficient bytes are available.
<b>CP_CheckBytesInsufficientBytes</b>	The pipe has retrieved all bytes but those available are less than those requested.
<b>CP_CheckBytesAtEndOfStream</b>	The pipe has reached the end of the stream and no more bytes are available.
<b>CP_CheckBytesOutOfBuffers</b>	All read/write buffers are currently in use.

#### **9.4.3.4 CP\_EVENTTYPE**

The CP\_EVENTTYPE enumeration defines events a content pipe may send to its client via a registered ClientCallback function.

**Table 10.9.4: Content Pipe Event Types**

<b>Value</b>	<b>Description</b>
<code>CP_BytesAvailable</code>	Bytes requested in a CheckAvailableBytes call which were formally unavailable are now available. The iParam parameter of the callback contains the number of bytes currently available.
<code>CP_Overflow</code>	The pipe has more data than it has space to store. The iParam parameter of the callback is unused.
<code>CP_PipeDisconnected</code>	The pipe been disconnected. The iParam parameter of the callback is unused.

#### **9.4.4 CP\_PIPE\_TYPE Methods**

The CP\_PIPE\_TYPE structures includes the methods below expressed as function pointers. Since OpenMAX IL shares the content pipe definition with other APIs (e.g. OpenMAX AL), the content pipe methods return OpenKODE error codes. Thus CPResult may the value zero indicating success or one the values defined in Appendix B.

##### **9.4.4.1 Open**

The Open method opens the specified content stream with the specified access type:

```
CPresult (*Open)(  
    CPHandle* hContent,  
    CPstring szURI,  
    CP_ACCESSTYPE eAccess);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds. Relevant errors include: KD\_EACCES, KD\_EBADF, KD\_EHOSTUNREACH, KD EINVAL, KD\_EIO, KD\_EISDIR, KD\_EMFILE, and KD\_ENOENT.

The parameters are as follows:

<b>Parameter</b>	<b>Description</b>
<code>hContent</code>	Pointer receiving the new content handle corresponding to the specified URI opened with the specified access type.
{out}	
<code>szURI</code>	URI specifying the location of the content.
{in}	
<code>eAccess</code>	Desired access to the content.
{in}	

#### **9.4.4.2 Close**

The `Close` method closes the specified content handle:

```
CPresult (*Close)(  
    CPHandle hContent);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.  
Relevant errors include: `KD_EINVAL`, and `KD_EIO`.

The parameters are as follows.

Parameter	Description
<code>hContent</code>	Content handle to be closed. [in]

#### **9.4.4.3 Create**

The `Create` method creates the specified content stream and returns a handle to it:

```
CPresult (*Create)(  
    CPHandle* hContent,  
    CPstring szURI);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.  
Relevant errors include: `KD_EACCES`, `KD_EBADF`, `KD_EHOSTUNREACH`, `KD_EINVAL`,  
`KD_EIO`, `KD_EISDIR`, `KD_EMFILE`, `KD_ENOENT`, and `KD_EEXIST`.

The parameters are as follows.

Parameter	Description
<code>hContent</code>	Pointer receiving the new content handle corresponding to the specified URI opened with the specified access type. [out]
<code>szURI</code>	URI specifying the desired location of the content. [in]

#### **9.4.4.4 CheckAvailableBytes**

The `CheckAvailableBytes` method verifies that the specified number of bytes are available for reading or writing depending on access type:

```
CPresult (*CheckAvailableBytes)(  
    CPHandle hContent,  
    Cpuint nBytesRequested,  
    CP_CHECKBYTESRESULTTYPE *eResult);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.  
Relevant errors include: `KD_EINVAL`, and `KD_EIO`.

The parameters are as follows.

<b>Parameter</b>	<b>Description</b>
<i>hContent</i>	Handle of content to check. [in]
<i>nBytesRequested</i>	The desired number of bytes. Result will depend on whether this number of bytes is actually available currently. [in]
<i>eResult</i>	Result of check (see definition of <code>EP_CHECKDYTERESULTTYPE</code> ). [out]

#### 9.4.4.5 SetPosition

The `SetPosition` method moves the pipe's byte position within a piece of content to the specified location.

```
CPresult (*SetPosition)(  
    CPhandle hContent,  
    CPint nOffset,  
    CP_ORIGINTYPE eOrigin);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds although returning from this function does not necessarily imply data from the new position is immediately available. Relevant errors include: `KD EINVAL`, and `KD EIO`.

The parameters are as follows.

<b>Parameter</b>	<b>Description</b>
<i>hContent</i>	Handle of content. [in]
<i>nOffset</i>	Offset of desired byte position relative to the specified origin. [in]
<i>eOrigin</i>	Origin from relative to which the offset applies. [in]

#### 9.4.4.6 GetPosition

The `GetPosition` method returns the pipe's byte position within a piece of content.

```
CPresult (*GetPosition)(  
    CPhandle hContent,  
    CPuint *pPosition);
```

This is a blocking call. The pipe should return from this call within 5 milliseconds. Relevant errors include: `KD EINVAL`, and `KD EIO`.

The parameters are as follows.

<b>Parameter</b>	<b>Description</b>
<i>hContent</i>	Handle of content. [in]

Parameter	Description
pPosition	Current byte position of the pipe within the specified content. [out]

#### 9.4.4.7 Read

The `Read` method retrieves data of the specified size from the content stream and advances the content pointer by the size of the data. Note that the pipe client provides the pointer to accept the data. This function is therefore appropriate for small high frequency reads.

```
CPresult (*Read)(
    CPhandle hContent,
    CPbyte *pData,
    CPuint nSize);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.

Relevant errors include: `KD_EINVAL`, and `KD_EIO`.

The parameters are as follows:

Parameter	Description
hContent	Handle of content. [in]
pData	Client specified pointer to receive data. [out]
nSize	Number of bytes to read. [in]

#### 9.4.4.8 ReadBuffer

The `ReadBuffer` method retrieves a buffer allocated by the pipe containing the requested number of bytes from the content stream. The content pointer advances by the number of bytes read. Note that the pipe itself provides the pointer to the data. This function is therefore appropriate for large low frequency reads. The client shall call `ReleaseReadBuffer` when done with the buffer to return it to the pipe.

In some cases the requested block might not reside in contiguous memory within the pipe implementation. For instance, if the pipe leverages a circular buffer then the requested block might straddle the boundary of the circular buffer. By default a pipe implementation performs a copy in this case to provide the block to the pipe client in one contiguous buffer. If, however, the client sets `bForbidCopy`, then the pipe returns only those bytes preceding the memory boundary. Here the client may retrieve the data in segments over successive calls.

```
CPresult (*ReadBuffer)(
    CPhandle hContent,
    CPbyte **ppBuffer,
    CPuint *nSize);
```

```
____CPbool bForbidCopy);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.

Relevant errors include: `KD_EINVAL`, and `KD_EIO`.

The parameters are as follows.

**Parameter**

**Description**

`#Content`

Handle of content.

`[in]`

`ppBuffer`

Pointer to receive pipe supplied data buffer.

`[out]`

`nSize`

Prior to call: number of bytes to read. After call: number of bytes

`[in/out]`

actually read.

`bForbidCopy`

If set the pipe shall never perform a copy opting instead to provide less bytes than in requested.

`[in]`

#### 9.4.4.9 ReleaseReadBuffer

The `ReleaseReadBuffer` returns a buffer previously acquired via a `ReadBuffer`.

```
CPresult (*ReleaseReadBuffer)(  
____CPhandle hContent,  
____CPbyte *pBuffer);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.

Relevant errors include: `KD_EINVAL`, and `KD_EIO`.

The parameters are as follows.

**Parameter**

**Description**

`#Content`

Handle of content.

`[in]`

`pBuffer`

Pipe supplied read buffer being released (i.e. returned to pipe).

`[in]`

#### 9.4.4.10 Write

The `Write` method writes data of the specified size to the content stream and advances the content pointer by the size of the data. Note that the pipe client provides the pointer to accept the data. This function is therefore appropriate for small high frequency writes.

```
CPresult (*Write)(  
____CPhandle hContent,  
____CPbyte *data,  
____CPuint nSize);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.

Relevant errors include: `KD_EINVAL`, `KD_EACCES`, and `KD_EIO`.

The parameters are as follows:

Parameter	Description
<i>hContent</i>	Handle of content. [in]
<i>pData</i>	Client specified pointer to data. [out]
<i>nSize</i>	Number of bytes to write. [in]

#### 9.4.4.11 GetWriteBuffer

The `GetWriteBuffer` method acquires a buffer allocated by the pipe corresponding to the next set of bytes to be written to the content and of the specified size. Note that the pipe itself provides the pointer to the data. This function is therefore appropriate for large low frequency writes. The client shall call `WriteBuffer` when done with the buffer to commit the write and return the buffer to the pipe.

```
CPresult (*GetWriteBuffer)(  
    CPhandle hContent,  
    CPbyte **ppBuffer,  
    CPuint *nSize);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.

Relevant errors include: `KD EINVAL`, `KD_EACCES`, and `KD_EIO`.

The parameters are as follows:

Parameter	Description
<i>hContent</i>	Handle of content. [in]
<i>ppBuffer</i>	Pointer to receive pipe supplied data buffer. [out]
<i>nSize</i>	Size of requested write buffer in bytes. [in]

#### 9.4.4.12 WriteBuffer

The `WriteBuffer` method commits a write buffer previously acquired via a `GetWriteBuffer`, returns the write buffer to the pipe, and advances the write pointer by the size of the committed data.

```
CPresult (*WriteBuffer)(  
    CPhandle hContent,  
    CPbyte *pBuffer,  
    CPuint nFilledSize);
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.

Relevant errors include: `KD EINVAL`, `KD_EACCES`, and `KD_EIO`.

The parameters are as follows.

Parameter	Description
<i>hContent</i>	Handle of content. [in]
<i>pBuffer</i>	Pipe supplied data buffer containing data to commit. [in]
<i>nFilledSize</i>	Size of actual bytes to commit (from beginning of buffer). [in]

```
/* Register a per handle client callback with the content pipe. */
CPresult (*RegisterCallback)( CPhandle hContent, CPresult
(*ClientCallback)(CP_EVENTTYPE eEvent, CPuint iParam));
```

#### 9.4.4.13 RegisterCallback

The `RegisterCallback` method registers an client event callback for a given content handle with the pipe.

```
CPresult (*RegisterCallback)(
    CPhandle hContent,
    CPresult (*ClientCallback)(
        CP_EVENTTYPE eEvent,
        CPuint iParam));
```

This is a blocking call. The pipe should return from this call within 20 milliseconds.

Relevant errors include: `KD EINVAL`, and `KD EIO`.

The parameters are as follows.

Parameter	Description
<i>hContent</i>	Handle of content. [in]
<i>ClientCallback</i>	Event callback to register. [in]

### 9.5 Acquiring a Content Pipe

We define a content pipe is defined as an interface so that more than one pipe may be implemented on a system and so that pipes may be passed from one part of the system to another. A media processing object uses a pipe as a mechanism to access content (as identified via a URI). The media processing object acquires a content pipe either through the system or from the client.

### **9.5.1 Indexes**

The header OMX\_Index.h contains the enumeration OMX\_INDEXTYPE, which contains all standard index values used with the functions OMX\_GetParameter, OMX\_SetParameter. Table 10-9 Table 9-5 describes the index values that relate to Content Pipes.

**Table 10-9-5: Index Values for Content Pipe**

<b>Index</b>	<b>Description</b>
OMX_IndexParamContentURI	OMX_PARAM_CONTENTURITYPE. Specifies the content by URI.
OMX_IndexParamCustomContentPipe	OMX_PARAM_CONTENTPIPETYPE. Specifies the client content pipe.

### **9.5.2 OMX\_PARAM\_CONTENTURITYPE**

An OpenMAX IL component parameter which specifies the URI of a component's target content.

OMX\_PARAM\_CONTENTURITYPE is defined as follows:

```
typedef struct OMX_PARAM_CONTENTURITYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U8 contentURI[1];
} OMX_PARAM_CONTENTURITYPE;
```

#### **9.5.2.1 Parameters**

The parameters for OMX\_PARAM\_CONTENTURITYPE are defined as follows:

- contentURI specifies the URI name, including any terminating bytes(s).

Note: The nSize parameters indicates the total size of the structure including the size of the contentURI parameter.

### **9.5.3 OMX\_PARAM\_CONTENTPIPETYPE**

An OpenMAX IL component parameter which specifies the content pipe used by the component to access content.

OMX\_PARAM\_CONTENTPIPETYPE is defined as follows:

```
typedef struct OMX_PARAM_CONTENTPIPETYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_HANDLETYPE hPipe;
} OMX_PARAM_CONTENTPIPETYPE;
```

### **9.5.3.1 Parameters**

The parameters for `OMX_PARAM_CONTENTPIPETYPE` are defined as follows.

- `hPipe` specifies handle of the custom content pipe.

### **9.5.4 Acquiring a Content Pipe from the IL Core**

A OpenMAX IL Core method for acquiring a content pipe for a given URI. A component that requires a content pipe should always retrieve the pipe via this method unless the OpenMAX IL Client overrides the content pipe with a `OMX_SetParameter` call using the `OMX_IndexParamCustomContentPipe` parameter.

`FillBufferDone` is defined as follows:

```
OMX_ERRORTYPE OMX_GetContentPipe(  
    OMX_OUT OMX_HANDLETYPE *hPipe,  
    OMX_IN OMX_STRING szURI)
```

The parameters are as follows:

Parameter	Description
<code>hPipe</code>	The handle of the content pipe being retrieved. <i>[out]</i>
<code>szURI</code>	A URI string that associates the content pipe to be retrieved. <i>[in]</i>

### **9.5.5 Content Pipe Related Errors**

A set of OpenMAX IL error codes dedicated to failures associated with accessing content:

Error	Description
<code>OMX_ErrorContentPipeOpenFailed</code>	The content pipe open operation failed.
<code>OMX_ErrorContentPipeCreationFailed</code>	The content pipe creation operation failed.

## **9.6 Example Use Cases**

### **9.6.1 Playback/Parser Use Case:**

Consider the playback use case where a media processing object is responsible for parsing data from piece of source content. The following steps occur:

1. The client specifies the source content to the object, e.g. by URI.

2. The client *optionally* specifies the mechanism for accessing the source content (i.e. the content pipe) to the object.
3. If the client does not specify the content pipe to the object, the object must acquire a pipe itself (e.g. via an OpenMAX IL Core function or some implementation specific mechanism).
4. At the appropriate time the object opens the content specified by the client using the content pipe.
5. The object performs reads on the source content using the content pipe, parses that content, and plays it.
6. At the appropriate time the object closes the content using the content pipe.

#### **9.6.2 Recording/Combiner Use Case:**

Consider the recording use case where a media processing object is responsible for emitting final data (perhaps muxed and packaged by a “combiner”) to a piece of content. The following steps occur:

1. The client specifies the destination content to the object, e.g. by URL.
2. The client *optionally* specifies the mechanism for accessing the destination content (i.e. the content pipe) to the object.
3. If the client does not specify the content pipe to the object, the object must acquire a pipe itself (e.g. via an OpenMAX IL Core function or some implementation specific mechanism).
4. At the appropriate time the object opens the content specified by the client using the content pipe.
5. The object performs writes on the destination content using the content pipe sending muxed/packaged data to it after capture said data.
6. At the appropriate time the object closes the content using the content pipe.

## 10 Implementing Buffer Sharing

Buffer sharing is implemented on a tunnel within a component and is transparent to other components. The non-supplier port is unaware whether the supplier's component allocated the buffers itself or re-used buffers from another of its ports. Furthermore, the supplier is unaware of whether the non-supplier's component will re-use the buffers that the supplier provided.

A tunnel between any two ports represents a dependency between those ports. Buffer sharing extends that dependency so that all ports that share the same set of buffers form an implicit dependency chain. Exactly one port in that dependency chain allocates the buffers shared by all of them.

If a component chooses to share buffers, its implementation may fulfill the tunnels requirements by doing the following:

- Provide re-used buffers on some supplier ports.
- Account for the needs of shared ports when communicating buffer requirements on ports.
- Internally pass a buffer from an input port to an output port between an `OMX_EmptyThisBuffer` call and its corresponding `EmptyBufferDone` call.

OpenMAX IL defines external component semantics to be compatible with sharing, although it does not explicitly require that a component support sharing. This section discusses the implementation of those semantics in the context of buffer sharing. If no components are sharing buffers, the implementation reduces to a simpler set of steps and obligations.

### 10.1.1.1 Component Transition from Loaded to Idle State with Sharing

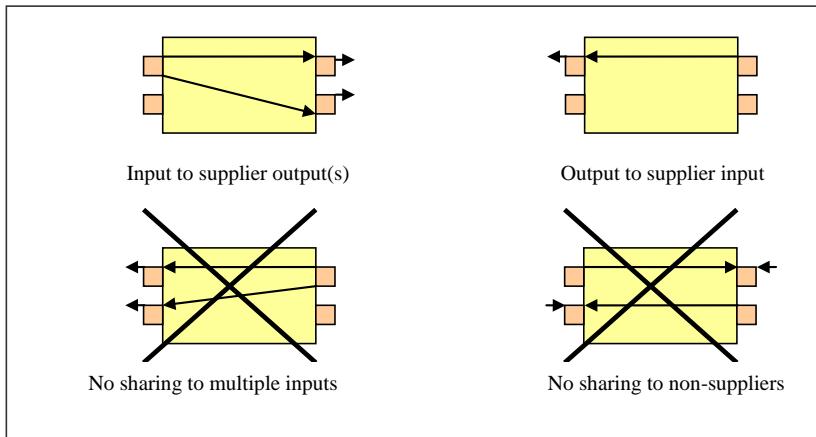
During the `OMX_SetupTunnel` call, the two ports of a tunnel establish which port (input or output) will act as the buffer supplier. Thus, when a component is commanded to transition from loaded to idle, it is aware of the roles of all its supplier or non-supplier ports.

When commanded to transition from loaded to idle, a component performs the following operations in this order:

1. The component determines what buffering sharing it will implement, if any.

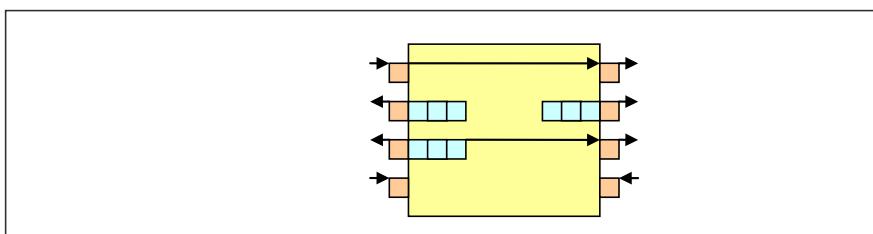
The following rules apply:

- a) A component may re-use a buffer only from one of its ~~one~~-input ports on one or more of its output ports or from one of its output ports on one of its input ports.
- b) Only a supplier port may re-use the buffers from another port.
- c) A component sharing buffers over multiple output ports requires read-only output port as shown in [Figure 10-1](#).



**Figure 10-1. Possible Sharing Relationships**

2. The component determines which of its supplier ports, if any, are also allocator ports. A supplier port is also an allocator port only if it does not re-use buffers from a non-supplier port on the same component (i.e., is not a sharing port).



**Figure 10-2. Determining Allocators:** a supplier port is a port with an arrow pointing away. A non-supplier port is a port with an arrow pointing toward it. An arrow from one port represents a sharing relationship. A port with boxes (buffers) adjacent to it represents an allocator port.

3. The component allocates its buffers for each of its allocator ports as follows:
  - For each port that re-uses the allocator ports buffer, the allocator port determines the buffer requirements of the sharing port. See obligation A below.
  - The allocator port determines the buffer requirements of its tunneled port via an OMX\_GetParameter call. See obligation B below.
  - The allocator port allocates buffers according to the maximum of its own requirements, the requirements of the tunneled port, and the requirement of all of the sharing ports.
  - The allocator port informs the tunneled non-supplier port that it is tunneling with of the actual number of buffers, via an

`OMX_SetParameter` call on `OMX_IndexParamPortDefinition` by setting the value of `nBufferCountActual` appropriately. See obligation E below.

- e) The allocator port shares its buffers with each sharing port that re-uses its buffers. See obligation D below.
- f) For every allocated buffer, the allocator port calls `OMX_UseBuffer` on its tunneling port after receiving the notification as defined in Section 3.1.3.13. See obligation C below.

A component shall also fulfill the following obligations:

- A. For a sharing port to determine its requirements, the sharing port shall first call `OMX_GetParameter` on its tunneled port to query for requirements and then return the maximum of its own requirements and the requirements of the tunneled ports. This request shall be made after receiving the `OMX_PORTSTATUS_ACCEPTUSEBUFFER` notification.
- B. When a non-supplier port receives an `OMX_GetParameter` call querying its buffer requirements, the non-supplier port shall first determine the requirements of all ports that re-use its buffers (see obligation A) and then return the maximum of its own requirements and those of its ports.
- C. When a non-supplier port receives an `OMX_UseBuffer` call from its tunneled port, the non-supplier port shall share the buffer with all ports on that component that re-use it.
- D. When a port A shares a buffer with a port B on the same component where port B re-uses the buffer of port A, then port B shall call `OMX_UseBuffer` and pass the buffer on its tunneled port.
- E. When a non-supplier port receives a `OMX_SetParameter` call on `OMX_IndexParamPortDefinition` from its tunneled port, the non-supplier port shall pass the `nBufferCountActual` field to any port that re-uses its buffers. Likewise, each supplier port that receives the `nBufferCountActual` field in this way shall pass the `nBufferCountActual` to its tunneled port by performing an `OMX_SetParameter` call on `OMX_IndexParamPortDefinition`. The actual number of buffers used throughout the dependency chain is propagated in this way.

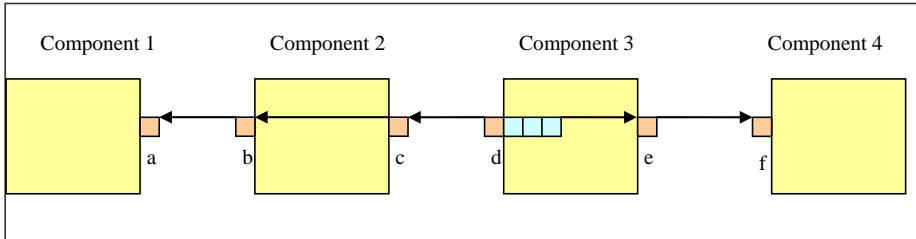
A component may transition from loaded to idle when all enabled ports have all the buffers they require.

In practice, there could be a direct mapping between the following:

- Steps 1–3 discussed earlier and code in the loaded-to-idle case in the state transition handler
- Obligation A and a subroutine to determine a shared ports buffer requirements
- Obligation B and the `OMX_GetParameter` implementation

- Obligation C and the `OMX_UseBuffer` implementation
- Obligation D and a subroutine to share a buffer from one port to another

To clarify why conformity to these steps and obligations leads to proper buffer allocation, consider the example illustrated in [Figure 10-3](#). Note that this example is contrived to exercise every step and obligation outlined above, and is therefore more complex than most real use cases.



**Figure 10-3. Example of Buffer Allocation with Sharing**

This discussion focuses only on the transition of component 3 to idle; similar operations occur inside the other components.

When the IL client commands component 3 to transition from loaded to idle, it follows the following prescribed steps:

1. Component 3 notices that it can re-use port d's buffers since port e is a supplier port. Component 3 establishes a sharing relationship from port d to port e.
2. Component 3 decides that since port d is a supplier port that does not re-use buffers, port d shall be an allocator port.
3. Component 3 allocates and distributes port d's buffers:
  - a) Since port e will re-use the buffer of port d, component 3 determines the buffer requirements of port e. In accordance with obligation A, port e calls `OMX_GetParameter` on port f to determine its buffer requirements and reports the requirements as the maximum between its own and those of port f.
  - b) Port d calls `OMX_GetParameter` on port c to determine its buffer requirements. In accordance with obligation B, port c shall determine the buffer requirements of port b. In accordance with obligation A, port b returns the maximum of its own requirements and the requirement of port a (retrieved via `OMX_GetParameter`) when queried. Port c then returns the maximum of its own requirements and the requirements that port b returns.
  - c) Port d allocates buffers according to the maximum of its own requirements and the requirements that ports c and e return. The resulting buffers are effectively allocated according to the maximum requirements of ports a, b, c, d, e, and f, all of which use the buffers of port d.

- d) Since port e will re-use the buffers of port d, component 3 shares these buffers with port e. In accordance with obligation D, port e calls `OMX_UseBuffer` on port f for every buffer that is shared.
- e) For each buffer allocated, port d calls `OMX_UseBuffer` on port c. In accordance with obligation C, port c shares each buffer with port b. Port b, in turn, obeys obligation D and calls `OMX_UseBuffer` on port a with the buffer.

Since all ports of all components now have their buffers, all components may transition to idle.

#### **10.1.1.2 Protocol for Using a Shared Buffer**

When an input port receives a shared buffer via an `OMX_EmptyThisBuffer` call, the input port may re-use that buffer on an output port *that it is sharing with the output port* by obeying the following rules:

- The output port calls `OMX_EmptyThisBuffer` on its tunneling port before the input port sends the corresponding `EmptyBufferDone` call to its tunneling port.
- The input port does not call `EmptyBufferDone` until all output ports on which the buffer is shared (i.e., via `OMX_EmptyThisBuffer` calls) return `EmptyBufferDone`.

## 11 Appendix A – References

This appendix identifies provides references to documentation on standards and formats presented in this document. The hyperlinks provide access to documents stored on various websites. The references are organized according to the applicable type of media.

### 11.1 SPEECH

#### 11.1.1 3GPP

- AMR-NB** [3G TS 26.071](#) "AMR speech Codec; General Description", Generation Partnership Project (3GPP). And references therein.
- AMR-WB** [3G TS 26.171](#) "AMR Wideband Speech Codec; General Description", Generation Partnership Project (3GPP). And references therein.
- GSM-EFR** [3G TS 46.051](#) "Enhanced Full Rate (EFR) speech processing functions; General description", Generation Partnership Project (3GPP). And references therein.
- GSM-FR** [3G TS 46.001](#) "Full rate speech; Processing functions", Generation Partnership Project (3GPP). And references therein.
- GSM-HR** [3G TS 46.002](#) "Half rate speech; Processing functions", Generation Partnership Project (3GPP). And references therein.

#### 11.1.2 3GPP2

- SMV** [3GPP2-SMV](#), "Selectable Mode Vocoder (SMV) Service Option for Wideband Spread Spectrum Communication Systems", 3GPP2 C.S0030-0, 2004.

#### 11.1.3 ARIB

- PDC-EFR** [RCR-27 EFR](#), "RCR-27-1: Personal Digital Cellular Telecommunication System," sec. 5.4, 2003.
- PDC-FR** [RCR-27 FR](#), "RCR-27-1: Personal Digital Cellular Telecommunication System," sec. 5.1, 2003.
- PDC-HR** [RCR-27 HR](#), "RCR-27-1: Personal Digital Cellular Telecommunication System," sec. 5.2, 2003.

#### 11.1.4 ITU

- G.711** [ITU-G711](#), "Pulse code modulation (PCM) of voice frequencies ", 1988.
- G.723.1** [ITU-G.723.1](#), "Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s", 1996.
- G.726** [ITU-G.726](#), "40, 32, 24, 16 kbit/s adaptive differential pulse code modulation (ADPCM)", 1990.

**G.729** [ITU-G.729](#), “Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear-prediction (CS-ACELP)”, 1996.

### 11.1.5 IETF

**RFC3267** [RFC3267](#): Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multirate Wideband (AMR WB) Audio Codecs.

### 11.1.6 TIA

**EVRC** [ANSI/TIA-127-A-2004](#), “Enhanced Variable Rate Codec Speech Service Option 3 for Wideband Spread Spectrum Digital Systems,” 2004.

**QCELP8** [ANSI/TIA/EIA-96-C-98](#), “Speech Service Option Standard for Wideband Spread Spectrum Systems,” 1998.

**QCELP13** [ANSI/TIA-733-A-2004](#), “High Rate Speech Service Option 17 for Wideband Spread Spectrum Communications Systems,” 2004.

**TDMA-EFR** [ANSI/TIA/EIA-136-410-1-2001](#), “TDMA Cellular PCS - Radio Interface - Enhanced Full-Rate Voice Codec, Addendum 1,” 2001.

**TDMA-FR** [ANSI/TIA/EIA-136-420-99](#), “TDMA Cellular PCS, VSELP,” 1999.

## 11.2 AUDIO

### 11.2.1 ISO

**HE-AAC v1** ISO/IEC JTC1/SC29/WG11 MPEG, [International Standard IS 14496-3 “Coding of Audio-Visual Objects—Part 3: Audio, Amendment 1: Bandwidth extension”, November 2003.](#)

**HE-AAC v2** ISO/IEC JTC1/SC29/WG11 MPEG, [International Standard IS 14496-3 “Coding of Audio-Visual Objects—Part 3: Audio, Amendment 2: Parametric coding for high-quality audio”, August 2004.](#)

**MPEG-1 Audio** 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.](#)

**MPEG-2 Audio** 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.](#)

**MPEG-2 AAC** ISO/IEC JTC1/SC29/WG11 MPEG, [International Standard IS 13818-7 “Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part 7: MPEG-2 AAC”, 2004.](#)

**MPEG-4 AAC** ISO/IEC JTC1/SC29/WG11 MPEG, [International Standard IS 14496-3 “Coding of Audio-Visual Objects—Part 3: Audio”, 2d Edition, December 2001.](#)

### 11.2.2 MISC

I3DL2	<a href="#">Interactive 3-D Audio Rendering Guidelines - Level 2.0</a> , Revision 1.0a. Interactive Audio Special Interest Group, September 20, 1999.
SBC	de Bont, F., Groenewegen, M., and Oomen, W., "A High Quality Audio Coding System at 128 kb/s", <a href="#">98th AES Convention</a> , Feb. 25-28, 1995.
WMA	<a href="#">Windows Media Audio</a>
VOR	<a href="#">Vorbis codec</a>
BIS	
RA	<a href="#">Real Audio 10 Codec</a>
PCM	<a href="#">Pulse-code Modulation</a>
ADPC	<a href="#">Adaptive Differential PCM</a>
M	
RFC 1766	Tags for the Identification of Languages ( <a href="http://www.ietf.org/rfc/rfc1766.txt">http://www.ietf.org/rfc/rfc1766.txt</a> )
ISO 639	Codes for the Representation of Names of Languages ( <a href="http://www.iso.org/iso/en/StandardsQueryFormHandler.StandardsQueryFormHandler?scope=CATALOGUE&amp;keyword=&amp;isoNumber=639&amp;sortOrder=ISO&amp;title=true&amp;search_type=ISO&amp;search_term=639&amp;languageCode=en">http://www.iso.org/iso/en/StandardsQueryFormHandler.StandardsQueryFormHandler?scope=CATALOGUE&amp;keyword=&amp;isoNumber=639&amp;sortOrder=ISO&amp;title=true&amp;search_type=ISO&amp;search_term=639&amp;languageCode=en</a> )
ISO 3166	Codes for the Representation of Names of Countries and their Subdivisions ( <a href="http://www.iso.org/iso/en/StandardsQueryFormHandler.StandardsQueryFormHandler?scope=CATALOGUE&amp;keyword=&amp;isoNumber=3166&amp;sortOrder=ISO&amp;title=true&amp;search_type=ISO&amp;search_term=3166&amp;languageCode=en">http://www.iso.org/iso/en/StandardsQueryFormHandler.StandardsQueryFormHandler?scope=CATALOGUE&amp;keyword=&amp;isoNumber=3166&amp;sortOrder=ISO&amp;title=true&amp;search_type=ISO&amp;search_term=3166&amp;languageCode=en</a> )

## 11.3 SYNTHETIC AUDIO

### 11.3.1 MIDI

DLS 1	<a href="#">Downloadable Sounds Level 1 Specification</a> , Version 1.1a, RP-016. MIDI Manufacturers Association, Los Angeles, CA, USA, January 1999.
DLS 2	<a href="#">Downloadable Sounds Level 2 Specification</a> , Version 1.0c, RP-025. MIDI Manufacturers Association, Los Angeles, CA, USA, July 14 1999. <a href="#">Downloadable Sounds Level 2.1 Specification</a> (RP-025/Amd1), MIDI Manufacturers Association, Los Angeles, CA, USA, January 2001.
General MIDI	<a href="#">The Complete MIDI 1.0 Detailed Specification, Document version 96.1</a> , 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)
General MIDI 2	<a href="#">General MIDI Level 2 Specification (Recommended Practice), v 1.1 (updated)</a> , RP-024. MIDI Manufacturers Association, Los Angeles, CA, USA, September 2003.

<b>GM Lite</b>	<a href="#">General MIDI Lite Specification and Guidelines for Use in Mobile Applications</a> , Version 1.0, RP-033. MIDI Manufacturers Association, Los Angeles, CA, USA, October 5, 2001.
<b>Mobile DLS</b>	<a href="#">Mobile DLS Specification, RP-041</a> , MIDI Manufacturers Association, Los Angeles, CA, USA, 2003.
<b>Mobile XMF (XMF type 2)</b>	<a href="#">Mobile XMF Content Format Specification, RP-042</a> . MIDI Manufacturers Association, Los Angeles, CA, USA, September 2004. <a href="#">XMF Meta File Format 2.0, RP-043</a> . MIDI Manufacturers Association, Los Angeles, CA, USA, September 2004. <a href="#">Scalable Polyphony MIDI Specification, Version 1.0, RP-034</a> . MIDI Manufacturers Association, Los Angeles, CA, USA, February 2002
<b>SP-MIDI</b>	<a href="#">Scalable Polyphony MIDI Device 5-24 Voice Profile for 3GPP, Version 1.0, RP-035</a> . MIDI Manufacturers Association, Los Angeles, CA, USA, February 2002. <a href="#">Type 0 and 1 XMF Files, RP-031</a> . MIDI Manufacturers Association, Los Angeles, CA, USA, 2001.
<b>XMF type 0 and 1</b>	<a href="#">XMF Meta File Format, Version 1.00b, RP-030</a> . MIDI Manufacturers Association, Los Angeles, CA, USA, October 2001. XMF Meta File Format Updates v1.01, RP-039. MIDI Manufacturers Association, Los Angeles, CA, USA, July 2003.

## 11.4 IMAGE

### 11.4.1 IETF

- RFC804** IETF/RFC 804, "[ITU Group 3 encoding: Modified Huffman and Modified Read compression algorithms](#)."
- RFC1314** IETF/RFC 1314, "[A File Format for the Exchange of Images in the Internet](#)," 1992.
- RFC2035** IETF/RFC 2305, "[RTP Payload Format for JPEG-compressed Video](#)," 1996.
- RFC2083** IETF/RFC 2083, "[PNG \(Portable Network Graphics\) Specification Version 1.0](#)," 1997.
- RFC2160** IETF/RFC 2160, "[Carrying PostScript in X.400 and MIME](#)," 1998.
- RFC2302** IETF/RFC 2302, "[Tag Image File Format \(TIFF\), image/tiff MIME Sub-type Registration](#)," 1998.
- RFC2306** IETF/RFC 2306, "[Tag Image File Format \(TIFF\), F Profile for Facsimile](#)," 1998.
- RFC3250** IETF/RFC 3250, "[Tag Image File Format Fax Extended \(TIFF-FX\), image/tiff-fx MIME Sub-type Registration](#)," 2002.
- RFC3302** IETF/RFC 3302, "[Tag Image File Format \(TIFF\) - image/tiff MIME Sub-type Registration](#)," 2002.
- RFC3362** IETF/RFC 3362, "[Real-time Facsimile \(T.38\), image/t38 MIME Sub-type Registration](#)," 2002.
- RFC3745** IETF/RFC 3745, "[MIME Type Registrations for JPEG 2000 \(ISO/IEC 15444\)](#)," 2004.

**RFC3950** IETF/RFC 3950, "[Tag Image File Format Fax Extended \(TIFF-FX\), image/tiff-fx MIME Sub-type Registration](#)," 2005.

#### 11.4.2 ISO

- JPEG v1** ISO/IEC JTC1/SC29/WG1 JPEG, International Standard IS 10918-1, "[Digital compression and coding of continuous-tone still images: Requirements and guidelines](#)," 1994.
- JPEG v2** ISO/IEC JTC1/SC29/WG1 JPEG, International Standard IS 10918-1/Cor 1, "[JPEG patent information update](#)," 2005.
- JPEG v3** ISO/IEC JTC1/SC29/WG1 JPEG, International Standard IS 10918-3, "[Digital compression and coding of continuous-tone still images: Extensions](#)," 1997.
- JPEG v4** ISO/IEC JTC1/SC29/WG1 JPEG, International Standard IS 10918-3/Amd 1, "[Provisions to allow registration of new compression types and versions in the SPIFF header](#)," 1999.
- JPEG v5** ISO/IEC JTC1/SC29/WG1 JPEG, International Standard IS 10918-4, "[Digital compression and coding of continuous-tone still images: Registration of JPEG profiles, SPIFF profiles, SPIFF tags, SPIFF colour spaces, APPmarkers, SPIFF compression types and Registration Authorities \(REGAUT\)](#)," 1999.
- JPEG v6** ISO/IEC JTC1/SC29/WG1 JPEG, International Standard IS 11544, "[Coded representation of picture and audio information, Progressive bi-level image compression](#)," 1993.
- JPEG LS v1** ISO/IEC JTC1/SC29/WG1 JPEG LS, International Standard IS 14495-1, "[Lossless and near-lossless compression of continuous-tone still images: Baseline](#)," 1999.
- JPEG LS v2** ISO/IEC JTC1/SC29/WG1 JPEG LS, International Standard IS 14495-2, "[Lossless and near-lossless compression of continuous-tone still images: Extensions](#)," 2003.
- JPEG 2000 v1** ISO/IEC JTC1/SC29/WG1 JPEG 2000, International Standard IS 15444-1, "[JPEG 2000 image coding system: Core coding system](#)," Ed. 2, 2004.
- JPEG 2000 v2** ISO/IEC JTC1/SC29/WG1 JPEG 2000, International Standard IS 15444-2, "[JPEG 2000 image coding system: Extensions](#)," Ed. 1, 2004.
- JPEG 2000 v3** ISO/IEC JTC1/SC29/WG1 JPEG 2000, International Standard IS 15444-6, "[JPEG 2000 image coding system, Part 6: Compound image file format](#)," Ed. 1, 2003.
- JPEG 2000 v4** ISO/IEC JTC1/SC29/WG1 JPEG 2000, International Standard IS 15444-12, "[JPEG 2000 image coding system, Part 12: ISO base media file format](#)," Ed. 2, 2005.

#### 11.4.3 ITU

- T81** ITU-T T.81, "[Digital compression and coding of continuous-tone still images, Requirements and guidelines](#)," 1992.
- T82** ITU-T T.82, "[Coded representation of picture and audio information, Progressive bi-level image compression](#)," 1993.
- T84 v1** ITU-T T.84, "[Digital compression and coding of continuous-tone still images](#):

	<a href="#">Extensions</a> , 1996.
T84 v2	ITU-T T.84/Amd 1, " <a href="#">Provisions to allow registration of new compression types and versions in the SPIFF header</a> ," 1999.
T85	ITU-T T.85, " <a href="#">Application profile for Recommendation T.82, Progressive bi-level image compression (JBIG coding scheme) for facsimile apparatus</a> ," 1995.
T86	ITU-T T.86, " <a href="#">Digital compression and coding of continuous-tone still images: Registration of JPEG Profiles, SPIFF Profiles, SPIFF Tags, SPIFF colour Spaces, APPn Markers, SPIFF Compression types and Registration Authorities (REGAUT)</a> ," 1998.
T87	ITU-T T.87, " <a href="#">Lossless and near-lossless compression of continuous-tone still images, Baseline</a> ," 1998.
T88 v1	ITU-T T.88, " <a href="#">Coded representation of picture and audio information, Lossy/lossless coding of bi-level images</a> ," 2000.
T88 v2	ITU-T T.88/Amd 1, " <a href="#">Encoder</a> ," 2003.
T88 v3	ITU-T T.88/Amd 2, " <a href="#">Extension of adaptive templates for halftone coding</a> ," 2003.
T89	ITU-T T.89, " <a href="#">Application profiles for Recommendation T.88, Lossy/lossless coding of bi-level images (JBIG2) for facsimile</a> ," 2001.

#### 11.4.4 JEITA

EXIF	JEITA, Japanese Electronics and Information Technology Industries Association, " <a href="#">EXIF (Exchangeable Image File Format) 2.2</a> ," 2002.
------	---

#### 11.4.5 MIPI

CSI	MIPI Camera WG, " <a href="#">CSI 2.0 Protocol Specification v.0.41</a> ," 2005.
DSI	MIPI Display WG, " <a href="#">DSI Specification v.0.45</a> ," 2005.

#### 11.4.6 Miscellaneous

BMP	<a href="#">Microsoft Windows Bitmap (BMP) Format</a> .
GIF87A	GIF 87a, " <a href="#">Graphics Interchange Format, Version 87a</a> ," 1987.
GIF89A	GIF 89a, " <a href="#">Graphics Interchange Format, Version 89a</a> ," 1989.
TIFF	TIFF V.6.0, " <a href="#">Tagged Image File Format (TIFF) Specification, Version 6.0</a> ".
WebP	<a href="#">WebP Image Format</a>

#### 11.4.7 SMIA

SMIA CCP2	SMIA CCP2, " <a href="#">Compact Camera Port 2 (CCP2) Specification 1.0</a> ."
SMIA CCP2/ER1	SMIA 1.0 CCP2/ER1, " <a href="#">Errata, Part 2 CCP2 Specification</a> ."
SMIA FUNC	SMIA Functional, " <a href="#">Functional specification 1.0</a> ."

<b>SMIA FUNC/ER1</b>	SMIA Functional 1.0/ER1, " <a href="#">Errata for Part 1 Functional Specification.</a> "
<b>SMIA CHAR</b>	SMIA Characterisation 1.0/V.A, " <a href="#">Characterisation Specification 1.0, Rev A.</a> "
<b>SMIA SW/AP</b>	SMIA Software And Application 1.0, " <a href="#">Software And Application Specification 1.0.</a> "

#### **11.4.8 W3C**

<b>PNG</b>	Portable Network Graphics (PNG) Specification (Second Edition), " <a href="#">Computer graphics and image processing, Portable Network Graphics (PNG): Functional specification</a> ," 2003.
------------	--

### **11.5 VIDEO**

#### **11.5.1 3GPP**

<b>MBMS v1</b>	3GPP TS 26.346 "MBMS Protocols and Codecs," v.1.5.0.
<b>MBMS v2</b>	3GPP TS 22.146 "Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Service." v.6.6.0.

#### **11.5.2 AVS**

<b>AVS-M v1</b>	AVS-M: Part 6 Video-Mobility, Stage 1: MMS service
<b>AVS-M v2</b>	AVS-M: Part 6 Video-Mobility, Stage 2: Streaming and conversational services

#### **11.5.3 DLNA**

<b>HNv1.0</b>	DLNA HNv1.0, "Home Networked Device Interoperability Guidelines v1.0," 2004.
---------------	--

#### **11.5.4 ETSI**

<b>DVB-H v1</b>	ETSI EN 302 304 V.1.1.1, DEN/JTC-DVB-155, "Digital Video Broadcasting (DVB), Transmission System for Handheld Terminals (DVB-H)," 2004.
<b>DVB-H v2</b>	ETSI ETS 300 468, RE/JTC-DVB-18, "Digital Video Broadcasting (DVB), Specification for Service Information (SI) in DVB systems," 1997.
<b>DVB-H v3</b>	ETSI EN 301 192 V.1.4.1, REN/JTC-DVB-157, "Digital Video Broadcasting (DVB), DVB specification for data broadcasting," 2004.
<b>DVB-H v4</b>	ETSI TS 101 154 V.1.7.1, RTS/JTC-DVB-170, "Digital Video Broadcasting (DVB), Implementation guidelines for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream," 2005.
<b>DVB-H v5</b>	ETSI TS 101 154 V.1.5.1, RTS/JTC-DVB-122, "Digital Video Broadcasting (DVB), Implementation guidelines for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream," 2004.
<b>DVB-H v6</b>	ETSI TS 102 005 V.1.1.1, DTS/JTC-DVB-124, "Digital Video Broadcasting

(DVB), Specification for the use of video and audio coding in DVB services delivered directly over IP," 2005.

**DVB-H v7** ETSI TS 102 154 V.1.2.1, RTS/JTC-DVB-123, "Digital Video Broadcasting (DVB), Implementation guidelines for the use of Video and Audio Coding in Contribution and Primary Distribution Applications based on the MPEG-2 Transport Stream," 2004.

### 11.5.5 IETF

- RFC1889** IETF RFC 1889, "[RTP: A Transport Protocol for Real-Time Applications](#)," 1996.
- RFC2032** IETF RFC 2032, "[RTP Payload Format for H.261 Video Streams](#)," 1996.
- RFC2038** IETF RFC 2038, "[RTP Payload Format for MPEG1/MPEG2 Video](#)," 1996.
- RFC2190** IETF RFC 2190, "[RTP Payload Format for H.263 Video Streams](#)," 1997.
- RFC2250** IETF RFC 2250, "[RTP Payload Format for MPEG1/MPEG2 Video](#)," 1998.
- RFC2429** IETF RFC 2429, "[RTP Payload Format for the 1998 Version of ITU-T Rec. H.263 Video \(H.263+\)](#)," 1998.
- RFC2431** IETF RFC 2431, "[RTP Payload Format for BT.656 Video Encoding](#)," 1998.
- RFC2435** IETF/RFC 2435, "[RTP Payload Format for JPEG-compressed Video](#)," 1998.
- RFC3189** IETF RFC 3189, "[RTP Payload Format for DV \(IEC 61834\) Video](#)," 2002.
- RFC3497** IETF RFC 3497, "[RTP Payload Format for Society of Motion Picture and Television Engineers \(SMPTE\) 292M Video](#)," 2003.
- RFC3551** IETF RFC 3551, "[RTP Profile for Audio and Video Conferences with Minimal Control](#)," 2003.
- RFC3984** [IETF RFC 3984, "RTP Payload Format for H.264 Video"](#), 2005.
- RFC4629** IETF RFC [4629, "RTP Payload Format for H.263 Video"](#), 2007.

### 11.5.6 ISO

- MPEG-1 Visual** ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 11172-2, "[Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Part 2: Video](#)," Ed. 1, 1993.
- MPEG-2 Visual** ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 13818-2, "[Generic coding of moving pictures and associated audio information, Part 2: Video](#)," Ed. 2, 2000.
- MPEG-4 Visual v1** ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 14496-1/Amd 7, "[Use of AVC \(Advanced Video Coding\) in MPEG-4 systems](#)," Ed. 1, 2004.
- MPEG-4 Visual v2** ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 14496-2, "[Coding of audio-visual objects, Part 2: Visual](#)," Ed. 3, 2004.
- MPEG-4 Visual v3** ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 14496-10, "[Coding of audio-visual objects, Part 10: Advanced Video Coding](#)," Ed. 2, 2004.

<b>MPEG-4</b>	ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS 14496-15,
<b>Visual v4</b>	<a href="#">"Coding of audio-visual objects, Part 15: Advanced Video Coding (AVC) file format,"</a> Ed. 1, 2004.
<b>MPEG-21</b>	ISO/IEC JTC1/SC29/WG11 MPEG, International Standard IS TR 21000-1,
<b>Visual</b>	<a href="#">"Vision, Technologies and Strategy, Part 1,"</a> 2001.
<b>MJPEG-2000</b>	ISO/IEC JTC1/SC29/WG1 MJPEG, International Standard IS 15444-3,
<b>v1</b>	<a href="#">"JPEG 2000 image coding system, Part 3: Motion JPEG 2000,"</a> Ed. 1, 2002.
<b>MJPEG-2000</b>	ISO/IEC JTC1/SC29/WG1 MJPEG, International Standard IS 15444-3/Amd 2, <a href="#">"Motion JPEG 2000 derived from ISO base media file format,"</a> Ed. 1, 2003.
<b>v2</b>	

### **11.5.7 ITU**

<b>H.261</b>	ITU-T H.261, " <a href="#">Video codec for audiovisual services at p x 64 kbit/s,</a> " 1993.
<b>H.262</b>	ITU-T H.262, " <a href="#">Generic coding of moving pictures and associated audio information: Video,</a> " 2000.
<b>H.263</b>	ITU-T H.263, " <a href="#">Video coding for low bit rate communication,</a> " 2005.
<b>H.264</b>	ITU-T H.264, " <a href="#">Advanced video coding for generic audiovisual services,</a> " 2005.

### **11.5.8 MISC**

<b>RV</b>	<a href="#">Real Video 10 Codec</a>
<b>WMV</b>	<a href="#">Windows Media Video</a>
<b>VC1</b>	<a href="#">Society of Motion Picture and Television Engineers, "VC-1 Compressed Video Bitstream Format and Decoding Process", SMPTE 421M.</a>
<b>RFC4425</b>	<a href="#">RTP Payload Format for Video Codec 1 (VC-1).</a>
<b>VP8</b>	<a href="#">VP8 Video Codec</a>

## **11.6 JAVA**

### **11.6.1 Multimedia**

<b>JSR-135</b>	JCP/JSR-135: <a href="#">Mobile Media API 1.1</a> , 2003
<b>JSR-234</b>	JCP/JSR-234: <a href="#">Advanced Multimedia Supplements</a> , 2005

### **11.6.2 Broadcast**

<b>JSR-272</b>	JCP/JSR-272: <a href="#">Mobile Broadcast Service API for Handheld Terminals</a> , 2005
----------------	---

**12 Appendix B—OpenKODE Error Codes**

~~13 Since OpenMAX IL shares the content pipe definition with other APIs (e.g. OpenMAX AL), the content pipe methods return OpenKODE error codes. This appendix provides a reference of OpenKODE error codes.~~

**14**

**Formatted:** Heading 1, Don't keep with next,  
Don't adjust space between Latin and Asian text

**Table 12-1: OpenKODE Error Codes**

Value	Description
-------	-------------

Formatted: Heading 1, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ACCESS</code>	<b>Permission denied.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EADDRINUSE</code>	<b>Address in use.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

Value	Description
KD_EAGAIN	<p><b>Resource unavailable, try again.</b></p>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EBADF</code>	<b>Bad file descriptor.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EBUSY</code>	<b>Device or resource busy.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ECONNREFUSED</code>	<b>Connection refused.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ECONNRESET</code>	<b>Connection reset.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EDEADLK</code>	<b>Resource deadlock would occur.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EDESTADDRREQ</code>	<p><b>Destination address required.</b></p> <p>Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text</p>

<b>Value</b>	<b>Description</b>
<code>KD_ERANGE</code>	<b>Mathematics argument out of range.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

Value	Description
KD_EEXIST	File exists.

Annotations:

- Value: Formatted: Heading 1, Don't adjust space between Latin and Asian text
- Description: Formatted: Heading 1, Don't adjust space between Latin and Asian text
- Description: Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

Value	Description
KD_EPBIG	<b>File too large.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EHOSTUNREACH</code>	<b>Host is unreachable.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

Value	Description
KD EINVAL	Invalid argument.

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

Value	Description
KD_EIO	I/O error.

Annotations:

- Value: Formatted: Heading 1, Don't adjust space between Latin and Asian text
- Description: Formatted: Heading 1, Don't adjust space between Latin and Asian text
- I/O error.: Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EISCONN</code>	<b>Socket is connected.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

Value	Description
KD_EISDIR	<b>Is a directory.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EMFILE</code>	<b>Too many open files.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

Value	Description
<code>KD_ENAMETOOLONG</code>	<b>Filename too long.</b> <p>Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text</p>

<b>Value</b>	<b>Description</b>
<code>KD_ENOENT</code>	<b>No such file or directory.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ENOMEM</code>	<b>Not enough space.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ENOSPC</code>	<b>No space left on device.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ENOSYS</code>	<b>Function not supported.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

Value	Description
KD_ENOTCONN	<p><b>The socket is not connected.</b></p>

**Formatted:** Heading 1, Don't adjust space between Latin and Asian text

**Formatted:** Heading 1, Don't adjust space between Latin and Asian text

**Formatted:** Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EPERM</code>	<b>Operation not permitted.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ETIMEDOUT</code>	<b>Connection timed out.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_EILSEQ</code>	<b>Illegal byte sequence.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

15

~~16 OpenMAX IL 1.1.1 specified a number of  
OpenKODE error codes to be relevant that were not  
retained in version 1.0 of OpenKODE specification.  
These error codes have been deprecated in the 1.1.2  
version of this specification. These error codes are:~~

**Formatted:** Heading 1, Don't keep with next,  
Don't adjust space between Latin and Asian text

## ~~Table 12-1: Deprecated OpenKODE Error Codes~~

Value	Description
-------	-------------

Formatted: Heading 1, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ECONNABORTED</code>	<b>Connection aborted.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ENETDOWN</code>	<b>Network is down.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ENETUNREACH</code>	<b>Network unreachable.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

<b>Value</b>	<b>Description</b>
<code>KD_ENOTSUP</code>	<b>Not supported.</b>

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Don't adjust space between Latin and Asian text

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Don't adjust space between Latin and Asian text

