



cinemo



Cinemo Media Engine API

STRICTLY CONFIDENTIAL– PROVIDED UNDER NDA

for **HUMAX Automotive / Z101**

Version 1.164.0.101454 (July 12, 2021)

COPYRIGHT © 2010-2021 CINEMO GMBH. ALL RIGHTS RESERVED WORLDWIDE.

THE CONTENTS OF THIS DOCUMENT ARE STRICTLY CONFIDENTIAL. THIS DOCUMENT IS PROVIDED UNDER NON-DISCLOSURE AGREEMENT WITH BETWEEN CINEMO GMBH AND THE RECIPIENT. ANY REDISTRIBUTION OF THIS DOCUMENT, EITHER IN PARTS OR AS A WHOLE IS STRICTLY PROHIBITED WITHOUT PRIOR WRITTEN APPROVAL OF CINEMO GMBH.

THIS DOCUMENT CONTAINS PRELIMINARY DESIGNS AND IT DOES NOT REPRESENT A COMMITTED PROJECT, NEITHER IN A PART OR AS A WHOLE. CINEMO GMBH RESERVES THE RIGHT TO MAKE CHANGES AT ANY TIME WITHOUT FURTHER NOTICE.

ALL TRADEMARKS MENTIONED IN THIS DOCUMENT ARE THE PROPERTY OF THEIR RESPECTIVE OWNERS.

HUMAX AUTOMATIC
CONFIDENTIAL

Contents

1 Cinemo Media Engine™	30
1.1 Playback	31
1.1.1 Virtual File System	32
1.1.2 Navigators	32
1.1.3 Decoders	33
1.1.4 Audio and Video Renderers	33
1.1.5 Distributed Playback	33
1.2 Playback Example	34
1.2.1 Configure Cinemo	34
1.2.2 Create Playlist	34
1.2.3 Create Player	34
1.2.4 Monitor Player Events	34
1.2.5 Extract Metadata	35
1.3 Playback of Blu-ray Discs	36
1.3.1 Interaction Model	36
1.3.2 Fully Asynchronous Model	37
1.3.3 Disc Ejection	37
1.4 UPnP Discovery	38
1.4.1 Configure Cinemo	38
1.4.2 Open SSDP	38
1.4.3 Watch Changes	39
1.4.4 Cancel/Enable the VFS	39
1.5 Browse UPnP Media Servers	41
1.5.1 Configure Cinemo	41
1.5.2 Create Playlist	41
1.5.3 Displaying Items in a User Interface	42
1.5.4 Playing Tracks	43
1.6 USB Devices	44
1.6.1 USB Discovery	44
1.6.2 USB Detection	45
1.6.3 USB Device Lifecycle	47
1.6.4 Browsing and Playback	47
1.6.5 MTP	48
1.7 Apple Devices	52
1.7.1 USB Host Mode	52
1.7.2 Authentication	53

1.7.3	Browsing	53
1.7.4	Playback	55
1.7.5	Playback Control from the Apple Device	55
1.7.6	iAP Interface	56
1.7.7	Indexing iAP devices	56
1.7.8	Native HID support	57
1.7.9	EAP support	57
1.7.10	Native EAP support	58
1.8	CarPlay	60
1.8.1	Architecture	60
1.8.2	CarPlay Client	60
1.8.3	Audio Playback and Input	61
1.8.4	User Events	61
1.8.5	Location Information	61
1.8.6	Vehicle Status	61
1.8.7	CarPlay and iAP-2 Protocol	62
1.8.8	iAP and USB Host Mode	62
1.8.9	Detecting CarPlay support	63
1.9	Android Auto	64
1.9.1	Architecture	64
1.9.2	Android Auto Client	65
1.9.3	Audio Playback and Input	65
1.9.4	Input Events	65
1.9.5	Video display handling	66
1.9.6	Input handling	66
1.9.7	ICinemoAndroidAuto life cycle	67
1.10	Playback from Audio and Video capture devices	69
1.10.1	Video Capture Parameters	70
1.10.2	Audio Capture Parameters	70
1.10.3	Linux Capture Example	70
2	Cinemo Media Management Engine™	72
2.1	Content Directory Service	73
2.1.1	UPnP Extensions	73
2.1.2	Cinemo Interfaces	73
2.1.3	Collation	74
2.1.4	Alphabetic Index	74
2.2	Items and Containers	76
2.2.1	Object IDs	76
2.2.2	Fields	76
2.2.3	Accessing Objects and Fields	76
2.2.4	ICinemoPlaylist	78

2.2.5	ICinemoVFS	79
2.2.6	Change Notifications	80
2.3	Volumes, Volume Groups and Hierarchies	81
2.3.1	Volumes	81
2.3.2	Volume Groups	82
2.3.3	Hierarchies	83
2.4	Indexed Volumes	84
2.4.1	Pass 1 – Filenames	84
2.4.2	Pass 2 – Metadata Extraction	85
2.4.3	Pass 3 – Thumbnails	85
2.4.4	Indexing Status	86
2.4.5	Indexing Errors	87
2.5	Virtual Volumes	88
2.5.1	Unsupported Devices	89
2.5.2	Inter-Process Communication	89
2.5.3	Arbitrary Metadata	89
2.6	Content Recognition Services	91
2.6.1	Editing API	92
2.6.2	Journaling API	92
2.6.3	Indexing API	93
2.6.4	Implications	94
2.7	Query Language	95
2.7.1	Indexed Fields	95
2.7.2	Operators	96
2.7.3	Sorting	97
2.8	Extended Text Matching	100
2.8.1	Word Boundary Matching	100
2.8.2	Phonetic Matching	100
2.9	Fields	103
2.9.1	@id	103
2.9.2	@parentID	103
2.9.3	@refID	103
2.9.4	@volumeID	103
2.9.5	@childCount	103
2.9.6	@searchable	103
2.9.7	@restricted	103
2.9.8	upnp:class	104
2.9.9	upnp:searchClass	104
2.9.10	cinemo:order	104
2.9.11	cinemo:updateID	105
2.9.12	cinemo:name	105

2.9.13	res	105
2.9.14	cinemo:type	106
2.9.15	cinemo:time	106
2.9.16	res@size	106
2.9.17	upnp:albumArtURI	106
2.9.18	cinemo:audio	107
2.9.19	cinemo:video	107
2.9.20	cinemo:totalItems	108
2.9.21	cinemo:totalAudioItems	108
2.9.22	cinemo:totalVideoItems	108
2.9.23	cinemo:totalImageItems	108
2.9.24	cinemo:volumeMountPath	108
2.9.25	cinemo:volumeUUID	108
2.9.26	cinemo:volumeType	108
2.9.27	cinemo:volumeStatus	109
2.9.28	cinemo:volumeStatusFlags	109
2.9.29	cinemo:volumeDismountStatus	110
2.9.30	cinemo:volumeError	110
2.9.31	cinemo:volumeMounted	111
2.9.32	cinemo:volumeIndexerStatus	111
2.9.33	cinemo:volumeIndexerPaused	111
2.9.34	cinemo:volumeProgress	111
2.9.35	cinemo:volumeProgressLength	111
2.9.36	cinemo:volumeProgressStatus	111
2.10	Configuration	112
2.10.1	Classes	113
2.10.2	Fields	114
2.10.3	Volume Groups	116
2.10.4	Hierarchy Rules	117
2.10.5	Hierarchy Building	118
2.11	Locales	120
2.11.1	Single Locale Configuration	120
2.11.2	Multi Locale Configuration	120
3	Cinemo Interfaces	122
3.1	ICinemoUnknown	123
3.1.1	AddRef	124
3.1.2	Release	125
3.1.3	QueryInterface	126
3.2	ICinemoUTF8	127
3.2.1	Assign	128
3.2.2	Format	129

3.2.3	FormatMediaType	130
3.2.4	Clear	131
3.2.5	Ptr	132
3.2.6	Ptrz	132
3.2.7	Bytes	133
3.2.8	Chars	134
3.2.9	Cmp	135
3.2.10	Cmpi	135
3.2.11	Cmpn	135
3.2.12	Cmpni	135
3.2.13	Atoi	136
3.2.14	Atoi64	136
3.2.15	SetURLParameter	137
3.2.16	GetURLParameter	138
3.3	ICinemoAudioCorrelator	139
3.3.1	UpdateLiveStreamParams	141
3.4	ICinemoBlob	142
3.4.1	Assign	143
3.4.2	AssignStatic	144
3.4.3	Load	145
3.4.4	Clear	146
3.4.5	Resize	147
3.4.6	Ptr	148
3.4.7	Bytes	149
3.4.8	Crc32	150
3.5	ICinemoMetaRawText	151
3.5.1	Assign	152
3.5.2	Clear	153
3.5.3	Ptr	154
3.5.4	Bytes	155
3.5.5	Srctype	156
3.5.6	Hint	157
3.5.7	Bpc	158
3.6	ICinemoDataPort	159
3.6.1	Open	160
3.6.2	Close	161
3.6.3	Read	162
3.6.4	Write	163
3.6.5	Cancel	164
3.6.6	Enable	164
3.7	ICinemoEncoder	165

3.7.1	GetAudioDeviceName	166
3.7.2	GetVideoDeviceName	167
3.8	ICinemoFile	168
3.8.1	GetURL	169
3.8.2	GetContentType	170
3.8.3	GetCaps	171
3.8.4	GetType	172
3.8.5	GetSize	173
3.8.6	GetDuration	174
3.8.7	GetCacheHints	175
3.8.8	Open	176
3.8.9	Close	177
3.8.10	WakeDevice	178
3.8.11	Cancel	179
3.8.12	Enable	179
3.8.13	Read	180
3.8.14	Write	181
3.8.15	SetSize	182
3.8.16	Flush	183
3.8.17	Reconnect	184
3.9	ICinemoLogConfig	185
3.9.1	GetZoneID	186
3.9.2	SetZoneID	187
3.9.3	GetMDC	188
3.9.4	SetMDC	189
3.9.5	LogMessage	190
3.10	ICinemoLogAppender	191
3.10.1	Open	192
3.10.2	Open	194
3.10.3	Close	195
3.10.4	SetLevel	196
3.10.5	SetLayout	197
3.11	ICinemoLogger	198
3.11.1	Message	199
3.11.2	Message	200
3.12	ICinemoLiveStream	201
3.12.1	Write	205
3.12.2	WriteMetadata	206
3.12.3	WriteDiscontinuity	207
3.12.4	WriteFormatChange	208
3.12.5	SignalServiceChange	209

3.12.6 Drain	210
3.12.7 Flush	211
3.13 ICinemoProjection	212
3.13.1 WriteVideoFormat	213
3.13.2 WriteVideo	214
3.13.3 SetAudioParams	215
3.13.4 WriteAudioFormat	216
3.13.5 WriteAudio	217
3.13.6 FlushAudio	218
3.14 ICinemoMetapool	219
3.14.1 GetItemCount	220
3.14.2 GetItem	221
3.14.3 GetCursor	222
3.14.4 Read	223
3.14.5 Find	224
3.14.6 Find	225
3.14.7 GetText	226
3.14.8 GetBlob	227
3.14.9 GetUInt32	228
3.14.10 GetUInt64	228
3.14.11 AddItem	229
3.14.12 AddUTF8	231
3.14.13 AddUInt32	232
3.14.14 AddUInt64	232
3.14.15 AddPool	233
3.14.16 RemoveAll	234
3.14.17 Refresh	235
3.15 ICinemoMetapoolCursor	236
3.15.1 GetNextItem	237
3.16 ICinemoOption	238
3.16.1 SetOption	239
3.16.2 GetOption	240
3.16.3 GetOptionType	241
3.16.4 GetOptionFlag	243
3.16.5 GetOptionEnum	244
3.16.6 GetOptionID	245
3.16.7 SaveOptions	246
3.16.8 LoadOptions	247
3.16.9 ResetOptions	248
3.16.10 SetOptionBlob	249
3.16.11 GetOptionBlob	250

3.16.12 SetOptionCallback	251
3.16.13 GetOptionCallback	252
3.17 ICinemoConfig	253
3.17.1 GetVersion	254
3.17.2 GetDefaultFolders	255
3.17.3 SetLog	256
3.17.4 SetLogLevel	258
3.17.5 SetPluginDirectory	259
3.17.6 SetCustomDecoderFactory	260
3.17.7 RefreshNetworkInterfaces	261
3.18 ICinemoPlaylist	262
3.18.1 SetEventCallback	264
3.18.2 SetEventQueue	265
3.18.3 SetEventSourceID	266
3.18.4 SetName	267
3.18.5 SetRepeat	268
3.18.6 SetOrder	269
3.18.7 SetRandomSeed	271
3.18.8 GetName	272
3.18.9 GetOpenURL	273
3.18.10 GetRepeat	274
3.18.11 GetOrder	275
3.18.12 GetCount	276
3.18.13 GetIndex	277
3.18.14 GetTrack	278
3.18.15 GetIndexOfTrackOffset	279
3.18.16 GetTrackOffsetAtIndex	280
3.18.17 Open	281
3.18.18 Close	282
3.18.19 Cancel	283
3.18.20 Enable	283
3.18.21 AppendTrack	284
3.18.22 AppendTrackWithID	285
3.18.23 AppendSource	286
3.18.24 Remove	287
3.18.25 RemoveAll	288
3.18.26 Move	289
3.18.27 Find	290
3.18.28 GetMetapool	291
3.18.29 GetURL	293
3.18.30 Select	295

3.18.31 SelectTrack	298
3.18.32 InitPlayback	300
3.18.33 InitPlayback2	301
3.18.34 GetAlphaIndexCount	302
3.18.35 GetAlphaIndex	303
3.18.36 GetCandidateCount	304
3.18.37 GetCandidate	305
3.18.38 Freeze	306
3.18.39 AttachPlayer	307
3.18.40 AttachPlayer2	308
3.19 ICinemoPlayer	309
3.19.1 SetEventCallback	312
3.19.2 SetEventQueue	313
3.19.3 SetEventSourceID	314
3.19.4 SetGraphParams	315
3.19.5 SetAudioParams	316
3.19.6 GetAudioParams	318
3.19.7 SetVideoParams	319
3.19.8 GetVideoParams	322
3.19.9 SetDistributed	323
3.19.10 SetSpeed	324
3.19.11 SetScanTime	325
3.19.12 SetTimeEventsInterval	326
3.19.13 SetAudioCaptureCallback	327
3.19.14 OpenTrack	328
3.19.15 CloseTrack	329
3.19.16 PlayTrack	330
3.19.17 NextTrack	331
3.19.18 PreviousTrack	332
3.19.19 Play	333
3.19.20 Stop	334
3.19.21 Seek	335
3.19.22 SeekTitleChapter	336
3.19.23 ShowMenu	337
3.19.24 ResumeTitle	338
3.19.25 ReturnFromSubmenu	339
3.19.26 NextChapter	340
3.19.27 PreviousChapter	341
3.19.28 ReplayChapter	342
3.19.29 SetRepeatChapter	343
3.19.30 SetRepeatTitle	344

3.19.31	StepForward	345
3.19.32	StepBackward	346
3.19.33	SelectAudio	347
3.19.34	SelectAngle	348
3.19.35	SelectSubpicture	349
3.19.36	AddExternalSubtitle	350
3.19.37	SelectButton	352
3.19.38	ActionButton	353
3.19.39	SelectButtonRelative	354
3.19.40	SelectButtonPosition	355
3.19.41	ActionButtonPosition	356
3.19.42	AcceptParentalLevel	357
3.19.43	AcceptCMI	358
3.19.44	GetStatus	359
3.19.45	GetGraphStatus	361
3.19.46	GetQualityInfo	363
3.19.47	GetPosition	364
3.19.48	GetDuration	365
3.19.49	GetBuffered	366
3.19.50	GetAudio	367
3.19.51	GetAngle	368
3.19.52	GetSubpicture	369
3.19.53	GetMediaInfo	370
3.19.54	GetTitle	371
3.19.55	GetChapter	372
3.19.56	SetSessionData	373
3.19.57	GetSessionPool	374
3.19.58	InitMetapool	375
3.19.59	InitPlaylistMetapool	376
3.19.60	InitPainter	377
3.19.61	SaveState	378
3.19.62	RestoreState	379
3.19.63	SignalPlaylistChanged	380
3.19.64	TakeSnapshot	381
3.19.65	GetWindowDeviceName	382
3.19.66	GetTrackURL	383
3.19.67	GetTrackVFSAttributes	384
3.19.68	GetDistributedServerURL	385
3.20	ICinemoPlayer2	386
3.20.1	SetEventCallback	389
3.20.2	SetEventQueue	389

3.20.3	SetEventSourceID	389
3.20.4	SetGraphParams	389
3.20.5	SetAudioParams	389
3.20.6	GetAudioParams	389
3.20.7	SetVideoParams	389
3.20.8	GetVideoParams	389
3.20.9	SetDistributed	389
3.20.10	SetScanTime	390
3.20.11	SetTimeEventsInterval	391
3.20.12	OpenTrack	392
3.20.13	PlayTrack	393
3.20.14	CloseTrack	394
3.20.15	NextTrack	395
3.20.16	PreviousTrack	395
3.20.17	Play	396
3.20.18	Seek	397
3.20.19	SeekTitleChapter	398
3.20.20	ShowMenu	399
3.20.21	ResumeTitle	400
3.20.22	ReturnFromSubmenu	401
3.20.23	ReplayChapter	402
3.20.24	SetRepeatChapter	403
3.20.25	SetRepeatTitle	404
3.20.26	StepForward	405
3.20.27	StepBackward	406
3.20.28	SelectAudio	407
3.20.29	SelectAngle	407
3.20.30	SelectSubpicture	407
3.20.31	AddExternalSubtitle	407
3.20.32	GetStatus	407
3.20.33	GetGraphStatus	407
3.20.34	GetQualityInfo	407
3.20.35	GetPosition	407
3.20.36	GetDuration	407
3.20.37	GetBuffered	407
3.20.38	GetAudio	407
3.20.39	GetAngle	408
3.20.40	GetSubpicture	409
3.20.41	GetMediaInfo	410
3.20.42	GetTitle	411
3.20.43	GetTitleFlags	412

3.20.44	GetChapter	413
3.20.45	SetSessionData	414
3.20.46	GetSessionPool	414
3.20.47	InitMetapool	414
3.20.48	InitPlaylistMetapool	414
3.20.49	InitPainter	414
3.20.50	SaveState	415
3.20.51	RestoreState	416
3.20.52	SignalPlaylistChanged	417
3.20.53	TakeSnapshot	418
3.20.54	GetWindowDeviceName	419
3.20.55	SelectSubpictureStyle	420
3.20.56	SelectSecondaryAudio	421
3.20.57	SelectSecondaryVideo	422
3.20.58	GetSecondaryAudio	423
3.20.59	GetSecondaryVideo	424
3.20.60	PostKeyUserEvent	425
3.20.61	PostKeyEvent	431
3.20.62	PostMouseEvent	432
3.20.63	GetTrackURL	433
3.20.64	GetTrackVFSAttributes	434
3.20.65	GetDistributedServerURL	435
3.21	ICinemoControlPoint	436
3.21.1	ConnectTo	437
3.21.2	Cancel	438
3.21.3	Enable	438
3.21.4	GetRendererType	439
3.21.5	GetRendererCaps	440
3.21.6	IsPlayable	441
3.21.7	GetPlaylist	442
3.22	ICinemoTrackCopier	443
3.22.1	SetCopierCallback	445
3.22.2	SetCopierSourceID	446
3.22.3	SetCopierTarget	447
3.22.4	SetCopierTrackTarget	448
3.22.5	SetCopierTrackFormat	449
3.22.6	SetCopierTrackStatus	450
3.22.7	SetCopierTimeEventsInterval	451
3.22.8	GetCopierStatus	452
3.22.9	GetCopierTrackStatus	453
3.22.10	EnableCopier	455

3.22.11	DisableCopier	456
3.22.12	ResetCopier	457
3.23	ICinemoTrackProvider	458
3.23.1	AttachPlayer	460
3.23.2	DetachPlayer	461
3.23.3	IsValidTrack	462
3.23.4	GetNextTrack	463
3.23.5	GetPrevTrack	463
3.23.6	GetTrackInfo	464
3.23.7	GetRepeat	465
3.23.8	GetOrder	466
3.23.9	SetRepeatAsync	467
3.23.10	SetOrderAsync	468
3.24	ICinemoTrackSelector	469
3.24.1	Select	470
3.24.2	SelectChild	472
3.25	ICinemoPainter	474
3.25.1	Paint	475
3.25.2	SetUpdateCallback	476
3.26	ICinemoAudioCodec	477
3.26.1	Initialize	478
3.26.2	Write	479
3.26.3	Drain	480
3.26.4	Flush	481
3.26.5	SetEncoderConfig	482
3.26.6	GetEncoderInfo	483
3.27	ICinemoVFS	484
3.27.1	Open	485
3.27.2	OpenSource	487
3.27.3	Watch	488
3.27.4	GetAudioStream	489
3.27.5	GetVideoStream	490
3.27.6	GetSubpictureStream	491
3.27.7	GetMediaInfo	492
3.27.8	GetTitleInfo	493
3.27.9	GetChapterInfo	494
3.27.10	Cancel	495
3.27.11	Enable	495
3.27.12	SetThumbFormat	496
3.27.13	Close	497
3.28	ICinemoMM	498

3.28.1	CreateServer	500
3.28.2	ConnectToServer	501
3.28.3	GetServerURL	502
3.28.4	Cancel	503
3.28.5	Enable	503
3.28.6	CreateVirtualFileServer	504
3.28.7	AddVolumeGroup	506
3.28.8	AddIndexedVolume	508
3.28.9	AddVirtualVolume	510
3.28.10	AddVirtualFile	512
3.28.11	AddVirtualFolder	513
3.28.12	MountVolume	514
3.28.13	DismountVolume	515
3.28.14	ReindexVolume	516
3.28.15	MountVolumes	517
3.28.16	DismountVolumes	518
3.28.17	ReindexVolumes	519
3.28.18	EditNode	520
3.28.19	RemoveNode	521
3.28.20	PauseVolumeIndexer	522
3.28.21	BrowseMetadata	523
3.28.22	BrowseDirectChildren	524
3.28.23	Search	525
3.28.24	SetOption	526
3.28.25	GetOption	528
3.28.26	GetFieldMap	529
3.28.27	GetHierarchyNodes	530
3.28.28	GetParentNodes	531
3.28.29	StartVolumeEvents	532
3.28.30	StopVolumeEvents	535
3.28.31	StartNodeEvents	536
3.28.32	StopNodeEvents	539
3.28.33	StartIndexingEvents	540
3.28.34	StopIndexingEvents	542
3.28.35	RegisterQueryHandler	543
3.28.36	UnregisterQueryHandler	545
3.28.37	RegisterMetadataHandler	546
3.28.38	UnregisterMetadataHandler	548
3.28.39	SetCachedRevision	549
3.28.40	GetCachedRevision	550
3.29	ICinemoEventQueue	551

3.29.1	Read	552
3.29.2	Post	553
3.29.3	ReadCancel	554
3.29.4	ReadEnable	555
3.29.5	RemoveAll	556
3.30	ICinemoMuxer	557
3.30.1	Initialize	558
3.30.2	Finalize	560
3.30.3	AddTrack	561
3.30.4	DeliverSample	562
3.31	ICinemoDDP	563
3.31.1	CreateServer	564
3.31.2	GetServerURL	565
3.32	ICinemoWindowBitmap	566
3.32.1	Assign	567
3.32.2	AssignUpdate	568
3.32.3	AssignImage	569
3.32.4	SavelImage	570
3.32.5	Resize	571
3.32.6	GetSize	572
3.32.7	Premultiply	573
3.32.8	Lock	574
3.32.9	Unlock	575
3.32.10	DrawRect	576
3.32.11	DrawString	577
3.32.12	DrawBitmap	578
3.33	ICinemoWindowOverlay	579
3.33.1	SetVisible	580
3.33.2	SetCropping	581
3.33.3	SetScaling	582
3.33.4	Sync	583
3.34	ICinemoWindow	584
3.34.1	SetEventCallback	585
3.34.2	SetEventQueue	586
3.34.3	SetEventSourceID	587
3.34.4	SetWindowPos	588
3.34.5	SetScreenSaver	589
3.34.6	SetFullscreen	590
3.34.7	SetPaintFreeze	591
3.34.8	GetDeviceName	592
3.34.9	GetWindowDeviceName	593

3.34.10	GetClientRect	594
3.34.11	GetCaps	595
3.34.12	InitBitmap	596
3.34.13	InitOverlay	597
3.35	ICinemoMediaType	598
3.35.1	Serialize	599
3.35.2	Deserialize	600
3.35.3	Assign	601
3.35.4	Get	602
4	Cinemo Interfaces for Android Auto	603
4.1	General remarks for Android Auto	604
4.1.1	Endpoints	604
4.1.2	Compatibility with future Android Auto releases	605
4.1.3	Android Auto error codes reported by ICinemoPlayer	606
4.1.4	Callback functions for Android Auto	606
4.2	Factory Functions for Android Auto	607
4.2.1	CinemoCreateAndroidAuto	607
4.3	ICinemoAndroidAuto	608
4.3.1	Setup	609
4.3.2	SetCallbacks	612
4.3.3	SetExtraCallbacks	613
4.3.4	SetAudioFocus	615
4.3.5	SetNavigationFocus	616
4.3.6	ConfirmAudioPlaybackStart	617
4.3.7	SetAudioParamsMicrophone	618
4.3.8	SetAudioParamsGuidance	619
4.3.9	SetAudioParamsMedia	620
4.3.10	SetCallAvailability	621
4.3.11	DisconnectSession	622
4.3.12	Open	623
4.3.13	Close	624
4.3.14	GetDisplay	625
4.3.15	GetMetapool	626
4.4	ICinemoAndroidAutoBluetooth	627
4.4.1	OnReadyForPairing	628
4.4.2	RequestDelayedPairing	629
4.4.3	SendAuthData	630
4.5	ICinemoAndroidAutoDisplay	631
4.5.1	ReportKey	632
4.5.2	ReportKeyLongPress	633
4.5.3	ReportAbsolute	634

4.5.4	ReportRelative	635
4.5.5	ReportTouch	636
4.5.6	ReportTouchPad	638
4.5.7	SendKeyEvent	639
4.5.8	SendTouchscreenEvent	640
4.5.9	UpdateTouchPadSensitivity	641
4.5.10	SetVideoFocus	642
4.5.11	SetMargins	643
4.5.12	SetContentInsets	644
4.5.13	SetStableContentInsets	645
4.5.14	GetMargins	646
4.5.15	GetContentInsets	647
4.5.16	GetStableContentInsets	648
4.6	ICinemoAndroidAutoMediaBrowser	649
4.6.1	getNode	650
4.6.2	ReportAction	651
4.7	ICinemoAndroidAutoMediaPlayback	652
4.7.1	ReportAction	653
4.8	ICinemoAndroidAutoNavigation	654
4.8.1	Start	655
4.8.2	Stop	656
4.9	ICinemoAndroidAutoNotification	657
4.9.1	Ack	658
4.9.2	Send	659
4.10	ICinemoAndroidAutoPhone	660
4.10.1	ReportAction	661
4.11	ICinemoAndroidAutoSensor	662
4.11.1	ReportAccelerometer	664
4.11.2	ReportCompass	665
4.11.3	ReportCompass3D	666
4.11.4	ReportDoor	668
4.11.5	ReportDrivingStatus	669
4.11.6	ReportEnvironment	670
4.11.7	ReportFuel	671
4.11.8	ReportGear	672
4.11.9	ReportGpsSatellite	673
4.11.10	ReportGyroscope	675
4.11.11	ReportHvac	676
4.11.12	ReportLight	677
4.11.13	ReportLocation	679
4.11.14	ReportNightMode	681

4.11.15	ReportOdometer	682
4.11.16	ReportParkingBrake	683
4.11.17	ReportPassenger	684
4.11.18	ReportRPM	685
4.11.19	ReportSpeed	686
4.11.20	ReportTirePressure	687
4.11.21	ReportTollCardData	688
4.11.22	ReportError	689
4.12	ICinemoAndroidAutoVendorExtension	690
4.12.1	SendData	691
4.13	ICinemoAndroidAutoAudioCallbacks	692
4.13.1	OnAudioSinkFocusRequest	693
4.13.2	OnAudioSinkStartPlayback	694
4.13.3	OnAudioSinkStopPlayback	695
4.13.4	On AudioSourceMicrophoneRequest	696
4.14	ICinemoAndroidAutoBluetoothCallbacks	697
4.14.1	OnBluetoothAuthenticationResult	698
4.14.2	OnBluetoothPairingRequest	699
4.15	ICinemoAndroidAutoCoreCallbacks	701
4.15.1	OnBatteryStatusNotification	702
4.15.2	OnBeginSession	703
4.15.3	OnBugreportRequest	704
4.15.4	OnEndSession	705
4.15.5	OnNewConnection	706
4.15.6	OnServiceDiscoveryRequest	707
4.15.7	OnSessionEvent	709
4.15.8	OnNavigationFocusRequest	710
4.15.9	OnUnrecoverableError	711
4.15.10	OnVoiceSessionNotification	712
4.16	ICinemoAndroidAutoInputCallbacks	713
4.16.1	OnInputFeedback	714
4.17	ICinemoAndroidAutoMediaBrowserCallbacks	715
4.17.1	OnMediaBrowserListNode	716
4.17.2	OnMediaBrowserRootNode	717
4.17.3	OnMediaBrowserSongNode	718
4.17.4	OnMediaBrowserSourceNode	719
4.18	ICinemoAndroidAutoMediaPlaybackCallbacks	721
4.18.1	OnMediaPlaybackMetadata	722
4.18.2	OnMediaPlaybackStatus	723
4.19	ICinemoAndroidAutoNavigationCallbacks	724
4.19.1	OnNavigationCurrentPosition	725

4.19.2	OnNavigationState	726
4.19.3	OnNavigationStatus	727
4.20	ICinemoAndroidAutoNotificationCallbacks	728
4.20.1	OnNotificationAck	729
4.20.2	OnNotificationArrival	730
4.20.3	OnNotificationSubscriptionStatus	731
4.21	ICinemoAndroidAutoPhoneCallbacks	732
4.21.1	OnPhoneStatus	733
4.22	ICinemoAndroidAutoVendorExtensionCallbacks	735
4.22.1	OnVendorExtensionDataAvailable	736
4.23	ICinemoAndroidAutoVideoCallbacks	737
4.23.1	OnVideoConfig	738
4.23.2	OnVideoFocusRequest	739
4.23.3	OnVideoReady	740
4.23.4	OnVideoUiConfigChange	741
5	Cinemo Interfaces for Apple Devices	743
5.1	General remarks for Apple devices	744
5.1.1	Control of life cycle for a connection to an Apple device	744
5.1.2	IAP identification with ICinemolAP	745
5.1.3	Callback interface usage	745
5.1.4	Support for normal external accessory protocol: EAP	747
5.1.5	Browsing iAP-2 devices and MediaLibrary information flags	748
5.2	ICinemoAppleAuth	750
5.2.1	Open	751
5.2.2	GetSerialNumber	752
5.2.3	GetCertificate	753
5.2.4	GetChallengeResponse	754
5.3	ICinemolAP	755
5.3.1	Open	758
5.3.2	Close	768
5.3.3	IdentificationInformationUpdate	769
5.3.4	StartHID	770
5.3.5	SendHIDReport	771
5.3.6	StopHID	772
5.3.7	ResumeDefaultPlayback	773
5.3.8	RequestAppLaunch	774
5.3.9	EAPSessionStatus	775
5.3.10	EAPSessionRecv	777
5.3.11	EAPSessionPost	779
5.3.12	EAPSessionSend	780
5.3.13	SendLocationInformation	782

5.3.14	StartMediaLibraryInformation	783
5.3.15	StopMediaLibraryInformation	784
5.3.16	StartMediaLibraryUpdates	785
5.3.17	StopMediaLibraryUpdates	787
5.3.18	SetMediaLibraryStatus	788
5.3.19	SignalMediaLibraryQueryUpdate	789
5.3.20	SetMediaLibraryProgress	790
5.3.21	SendPlayMediaLibraryCommand	791
5.3.22	StartPowerUpdates	793
5.3.23	StopPowerUpdates	794
5.3.24	SendPowerSourceUpdate	795
5.3.25	StartCallStateUpdates	796
5.3.26	StopCallStateUpdates	797
5.3.27	StartCommunicationsUpdates	798
5.3.28	StopCommunicationsUpdates	799
5.3.29	StartListUpdates	800
5.3.30	StopListUpdates	801
5.3.31	InitiateCall	802
5.3.32	AcceptCall	803
5.3.33	EndCall	804
5.3.34	SwapCalls	805
5.3.35	MergeCalls	806
5.3.36	HoldStatusUpdate	807
5.3.37	MuteStatusUpdate	808
5.3.38	SendDTMF	809
5.3.39	SendVehicleStatus	810
5.3.40	BluetoothComponentInformation	811
5.3.41	StartBluetoothConnectionUpdates	812
5.3.42	StopBluetoothConnectionUpdates	813
5.3.43	RequestWiFiInformation	814
5.3.44	AccessoryWiFiConfigurationInformation	815
5.3.45	StartAssistiveTouch	816
5.3.46	StopAssistiveTouch	817
5.3.47	StartAssistiveTouchInformation	818
5.3.48	StopAssistiveTouchInformation	819
5.3.49	OOBBTParingAccessoryInformation	820
5.3.50	OOBBTPairingCompletionInformation	821
5.3.51	StartRouteGuidanceUpdates	822
5.3.52	StopRouteGuidanceUpdates	823
5.4	ICinemoAPEAPAnnounce	824
5.4.1	EAPSessionInitiated	825

5.4.2	EndEAPSession	826
5.4.3	EAPPacketArrived	827
5.5	ICinemolAPLocationInfo	828
5.5.1	StartLocationInformation	829
5.5.2	GPRMCDDataStatusValuesNotification	830
5.5.3	StopLocationInformation	831
5.6	ICinemolAPMediaItemReceiver	832
5.6.1	OnCollectionQueryItemCount	833
5.6.2	OnMediaItem	834
5.7	ICinemolAPMediaLibraryAccess	835
5.7.1	OnMediaLibraryInformation	836
5.7.2	OnMediaLibraryUpdate	837
5.7.3	GetMediaItem	839
5.7.4	GetCollectionListing	840
5.7.5	OnArtwork	842
5.7.6	OnStatus	843
5.7.7	StopWatching	844
5.8	ICinemolAPPowerUpdates	845
5.8.1	PowerUpdate	846
5.9	ICinemolAPCommunications	847
5.9.1	CallStateUpdate	848
5.9.2	CommunicationsUpdate	849
5.9.3	ListUpdate	850
5.10	ICinemolAPVehicleStatus	852
5.10.1	StartVehicleStatus	853
5.10.2	StopVehicleStatus	854
5.11	ICinemolAPBluetoothUpdates	855
5.11.1	BluetoothConnectionUpdate	856
5.11.2	InitialBluetoothComponentInformationRequest	857
5.12	ICinemolAPWiFilnfoSharing	858
5.12.1	WiFilnformation	859
5.12.2	RequestAccessoryWiFiConfigurationInformation	860
5.13	ICinemolAPAssistiveTouch	861
5.13.1	AssistiveTouchInformation	862
5.14	ICinemolAPOOBBTPairing	863
5.14.1	StartOOBBTPairing	864
5.14.2	OOBBTPairingLinkKeyInformation	865
5.14.3	StopOOBBTPairing	866
5.15	ICinemolAPVoiceOver	867
5.15.1	VoiceOverUpdate	868
5.15.2	VoiceOverCursorUpdate	869

5.16 ICinemolAPRouteGuidance	870
5.16.1 RouteGuidanceUpdate	871
5.16.2 RouteGuidanceManeuverUpdate	872
5.17 ICinemoCarPlay	873
5.17.1 Start	874
5.17.2 Stop	875
5.17.3 ForceKeyFrame	876
5.17.4 RequestSiri	877
5.17.5 SetLimitedUI	878
5.17.6 SetNightMode	879
5.17.7 RequestUI	880
5.17.8 SetCallbacks	881
5.17.9 SetInfo	883
5.17.10 SetMainAudioParams	885
5.17.11 SetAlternateAudioParams	885
5.17.12 SetInputAudioParams	885
5.17.13 ChangeModes	886
5.17.14 SendHIDReport	887
5.17.15 UpdateVehicleInformation	888
5.17.16 SendConnect	889
5.17.17 GetIAPUrlWifi	890
5.17.18 ReleaselapUrl	891
5.17.19 Disconnect	892
5.18 ICinemoCarPlay Callbacks	893
5.18.1 Initialize	894
5.18.2 Finalize	895
5.18.3 Started	896
5.18.4 ModesChanged	897
5.18.5 RequestUI	898
5.18.6 DisableBluetooth	899
5.18.7 SetInputModule	900
5.18.8 GetAuthenticationCertificate	901
5.18.9 GetAuthenticationSignature	902
5.18.10 AudioPrepare	903
5.18.11 AudioStop	905
5.18.12 AudioRampVolume	906
5.18.13 VideoPrepare	907
5.18.14 VideoStop	908
5.18.15 Heartbeat	909
5.18.16 ServiceUpdate	910
5.18.17 ConnectionResult	911

6 Cinemo Transport Library	912
6.1 General Remarks	913
6.1.1 Requirements	913
6.2 Transport Interface	914
6.2.1 Create_Transport_Custom	920
6.2.2 protocol_version	922
6.2.3 ctli_handle	923
6.2.4 ctli_delete_transport_fn	924
6.2.5 ctli_receive_fn	925
6.2.6 ctli_send_fn	926
6.2.7 ctli_abort_fn	927
6.2.8 ctli_get_param_fn	928
6.2.9 ctli_set_param_fn	930
6.2.10 ctli_audio_create_fn	931
6.2.11 ctli_audio_delete_fn	932
6.2.12 ctli_audio_receive_fn	933
6.2.13 ctli_audio_abort_fn	934
6.2.14 ctli_audio_get_params_fn	935
6.2.15 ctli_audio_set_params_fn	936
6.2.16 ctli_dataport_select_fn	937
6.2.17 ctli_dataport_send_fn	938
6.2.18 ctli_dataport_recv_fn	939
6.2.19 ctli_dataport_cancel_fn	940
6.2.20 ctli_dataport_enable_fn	941
6.2.21 ctli_dataport_get_param_fn	942
6.2.22 ctli_dataport_set_param_fn	943
6.3 Error Codes	944
6.3.1 CTLI_NO_ERROR	944
6.3.2 CTLI_ERROR_ARGUMENTS	944
6.3.3 CTLI_ERROR_AUDIOPARAMS	944
6.3.4 CTLI_ERROR_CANCEL	944
6.3.5 CTLI_ERROR_DECODE	944
6.3.6 CTLI_ERROR_IO	944
6.3.7 CTLI_ERROR_NOTCONNECTED	944
6.3.8 CTLI_ERROR_NOTENOUGHSPACE	944
6.3.9 CTLI_ERROR_NOTIMPLEMENTED	944
6.3.10 CTLI_ERROR_OPEN	945
6.3.11 CTLI_ERROR_OVERFLOW	945
6.3.12 CTLI_ERROR_RESOURCES	945
6.3.13 CTLI_ERROR_AUDIOSETUP	945
6.3.14 CTLI_ERROR_CONFIGURATION	945

6.3.15	CTLI_ERROR_IDLE	945
6.3.16	CTLI_ERROR_UNAVAILABLE	945
6.3.17	CTLI_ERROR_DRIVER_FAILURE	945
6.4	Cinemo SDK options	946
6.4.1	Transport Library Implementations	946
6.4.2	Transport Library Option String	946
7	Cinemo Apple Authentication Library	949
7.1	General Remarks	950
7.1.1	Requirements	950
7.2	Apple Authentication Interface	951
7.2.1	Create_AppleAuth_Custom	953
7.2.2	protocol_version	955
7.2.3	cali_handle	956
7.2.4	cali_delete_authentication_fn	957
7.2.5	cali_get_serial_number_fn	958
7.2.6	cali_get_certificate_fn	959
7.2.7	cali_get_challenge_response_fn	960
7.3	Error Codes	961
7.3.1	CALI_NO_ERROR	961
7.3.2	CALI_ERROR_ARGUMENTS	961
7.3.3	CALI_ERROR_IO	961
7.3.4	CALI_ERROR_NOTIMPLEMENTED	961
7.3.5	CALI_ERROR_OPEN	961
7.3.6	CALI_ERROR_OVERFLOW	961
7.3.7	CALI_ERROR_RESOURCES	961
7.4	Cinemo SDK options	962
7.4.1	Apple Authentication Library URL	962
7.4.2	Apple Authentication Library Implementations	962
7.4.3	Apple Authentication Library Option String	963
8	Cinemo Events	965
8.1	CINEMO_EC_OPEN	966
8.2	CINEMO_EC_OPEN_GAPLESS	967
8.3	CINEMO_EC_DOMAIN	968
8.4	CINEMO_EC_CUE	969
8.5	CINEMO_EC_PLAYSPEED	970
8.6	CINEMO_EC_TITLE	971
8.7	CINEMO_EC_CHAPTER	972
8.8	CINEMO_EC_ANGLE	973
8.9	CINEMO_EC_AUDIO	974
8.10	CINEMO_EC_SUBPICTURE	975

8.11 CINEMO_EC_BUTTON	976
8.12 CINEMO_EC_SUBPICTURE_RECT	977
8.13 CINEMO_EC_PROHIBITED_UOPS	978
8.14 CINEMO_EC_STILL	979
8.15 CINEMO_EC_ERROR	980
8.16 CINEMO_EC_WARNING	981
8.17 CINEMO_EC_PARENTAL_LEVEL	982
8.18 CINEMO_EC_TIME	983
8.19 CINEMO_EC_FINISHED	984
8.20 CINEMO_EC_METADATA	985
8.21 CINEMO_EC_PLAYLIST_METADATA	986
8.22 CINEMO_EC_PLAYLIST_CHANGED	987
8.23 CINEMO_EC_SESSION_DATA	988
8.24 CINEMO_EC_GRAPH_STATUS	989
8.25 CINEMO_EC_VIDEO_STATUS	990
8.26 CINEMO_EC_AUDIO_STATUS	991
8.27 CINEMO_EC_TRACK	992
8.28 CINEMO_EC_CLOSE	993
8.29 CINEMO_EC_STATUS	994
8.30 CINEMO_EC_POPUP_MENU	995
8.31 CINEMO_EC_SECONDARY_VIDEO	996
8.32 CINEMO_EC_SECONDARY_AUDIO	997
8.33 CINEMO_EC_SUBPICTURE_STYLE	998
8.34 CINEMO_EC_AUDIO_PROP_CHANGE	999
8.35 CINEMO_EC_VIDEO_PROP_CHANGE	1000
8.36 CINEMO_EC_TITLE_TRANSITION	1001
8.37 CINEMO_EC_CMI	1002
8.38 CINEMO_EC_EOF	1005
8.39 CINEMO_EC_OPERATION_ERROR	1006
8.40 CINEMO_EC_PLAYLIST_ORDER	1007
8.41 CINEMO_EC_PLAYLIST_REPEAT	1008
8.42 CINEMO_EC_PREGAP	1009
8.43 CINEMO_EC_SELECT	1010
8.44 CINEMO_EC_AUDIO_WATERMARK_MUTE	1011
8.45 CINEMO_EC_KEYFRAME_REQUEST	1012
8.46 CINEMO_EC_PLAYLIST_EMPTY	1013
8.47 CINEMO_EC_REMOTE_VOLUME	1014
8.48 CINEMO_EC_CORRELATION	1015
8.49 CINEMO_EC_VIDEO_VIEWPORT	1016
8.50 CINEMO_EC_WINDOW_RENDERCONFIG	1017
8.51 CINEMO_EC_REPEAT CHAPTER	1018

8.52 CINEMO_EC_REPEAT_TITLE	1019
8.53 CINEMO_EC_HTTP	1020
8.54 CINEMO_EC_INTERNAL	1021
9 Cinemo Error Codes	1022
9.1 Success Codes	1022
9.2 Failure Codes	1022
10 Cinemo Media Types	1031
10.1 Media Types	1032
10.2 Media Subtypes	1033
10.2.1 Audio Subtypes	1033
10.2.2 Video Subtypes	1035
10.2.3 Image Subtypes	1036
10.2.4 Uncompressed Subtypes	1036
10.2.5 Subtitle Subtypes	1037
10.3 Media Format	1038
10.3.1 Cinemo Video Format	1039
10.3.2 Cinemo Audio Format	1042
11 Cinemo Configuration Options	1043
11.1 General Options	1044
11.2 HTTP Options	1057
11.3 SSDP Options	1061
11.4 Protection Options	1063
11.5 Video Options	1066
11.6 Audio Options	1073
11.7 Distributed Playback Options	1086
11.8 Plugin Options	1091
11.9 ATAPI Options	1095
11.10 DVD Options	1098
11.11 Subtitle Options	1101
11.12 Thread Priority Options	1105
11.13 Thread Scheduling Policy Options	1107
11.14 Media Management Options	1109
11.15 Apple Authentication Options	1129
11.16 IAP Options	1130
11.17 USB Options	1134
11.18 DLNA Options	1135
11.19 Miscellaneous Options	1136
12 Cinemo Logging	1137
12.1 Targets for Log output	1138

12.2 XML File Based Configuration	1138
13 Cinemo Sample Applications	1139
13.1 SampleAudioCodec	1140
13.1.1 Decoding Usage	1140
13.1.2 Encoding Usage	1140
13.2 SampleBrowse	1141
13.3 SampleCopier	1142
13.4 SampleMetadata	1143
13.5 SampleMM	1144
13.6 SampleMMPhonetic	1146
13.7 SampleMMRemoteIndexing	1147
13.8 SamplePlayer	1148
13.9 SamplePlaylistSelect	1149
13.9.1 Example for local browsing	1151
13.9.2 Example for remote browsing	1153
13.10 SampleDDP	1155
13.11 SampleWindowBitmap	1156
14 Cinemo Tutorials	1157
14.1 AndroidAuto	1158
14.1.1 Connecting a VEC session remotely	1158
14.2 Apple Devices	1160
14.2.1 Searching iAP1 devices	1160
14.2.2 Highlighting the currently playing track	1163
14.2.3 Remote access for the Apple Authentication chip	1164
14.3 DDP	1165
14.3.1 DDP Server	1165
14.3.2 DDP Client	1165
14.3.3 Apple Devices	1166
14.3.4 DDP Security	1167
14.4 Distributed Playback	1168
14.4.1 Discovery	1168
14.4.2 Playback	1168
14.5 MediaServer	1169
14.5.1 Starting a Media Server	1169
14.5.2 Discovering a Media Server	1169
14.5.3 Browsing a Media Server	1169
14.5.4 Playing Media from a Media Server	1170
14.5.5 Indexing Volumes from a Media Server	1170
Appendix A Supported Formats	1172
A.1 File Navigators and Containers	1172

A.2	Playlist Formats	1172
A.3	Disc Navigators	1173
A.4	Audio Decoders	1173
A.5	Audio Encoders	1176
A.6	Video Decoders	1176
A.7	Image Decoders	1177

1. Cinemo Media Engine™

Cinemo Media Engine is an embedded multimedia playback stack, designed to run on multiple embedded operating systems including Linux, Android, QNX, Windows Embedded and Windows CE. It consists of a modular high-speed architecture and is entirely self-contained, with all parsers, navigators, decoders and renderers built-in. Video hardware acceleration is supported on a wide range of target platforms and devices.

Cinemo Media Engine is delivered as an SDK. It is optimized for each target platform and delivered as a set of pre-compiled libraries, C/C++ header files and sample source code. It is important to note that Cinemo does not run as a process, daemon, or any kind of executable application (except for those applications which may be provided for demonstration purposes). Instead, customers build their own applications using Cinemo SDK. Customers maintain full control of their target system, including all Cinemo objects, interfaces, and their instantiated lifetimes.

Cinemo Media Engine functionality is exposed by a high-level API which is designed to fulfill the common requirements of any application wishing to support media playback of one kind or another:

- The need to play media content.
- The need to display metadata extracted from the media content.
- The need to locate (or browse to) media content.

Note that these requirements do not include indexing, searching, filtering or sorting, which are handled separately by Cinemo Media Management Engine™.

The above list of requirements are distilled by Cinemo API, since the latter two requirements can be reduced to essentially the same function: *metadata extraction*. In the first case, applications need to obtain metadata about media items, such as MP3 files, while in the second case, applications are able to locate or browse media items if they can obtain metadata about the children of containers, such as file system folders, in which the items are present. Cinemo Media Engine is divided conceptually into the following functions:

- *Playback*, exposed by [ICinemoPlayer](#) interface and which offers playback, control, and event notifications including 'now playing' metadata.
- *Metadata Extraction*, exposed by [ICinemoVFS](#) and [ICinemoPlaylist](#) interfaces, and which offer, independently of playback, extraction of metadata from both media content items and their parent containers.

Cinemo imposes no restrictions on the use of these functions. All instantiated Cinemo objects are independent from each other. An application may instantiate, for example, multiple [ICinemoPlayer](#) objects to play multiple tracks at the same time. It may also instantiate multiple [ICinemoVFS](#) objects to simultaneously extract metadata. Cinemo interfaces should be used within reason – for example, if the source media is slow, such as an optical disc, it is not advisable to both play a track and simultaneously extract metadata from the same or

other tracks on the same optical media. All functions will succeed, but playback may stutter. Cinemo does offer options such as configurable thread priorities which may be used to alleviate such symptoms, but it is recommended that calling applications implement higher level system management logic to avoid such scenarios. Cinemo Media Engine is not responsible for system management at this level.

1.1 Playback

Cinemo supports playback of a wide variety of media content and formats, including but not limited to regular files on mass storage devices, optical formats such as DVD and blu-ray, USB protocols such as iAP and MTP, and network protocols such as HTTP and UPnP. A complete list of supported formats, protocols and audio and video codecs can be obtained from your Cinemo representative.

Playback functionality is exposed by [ICinemoPlayer](#) interface. This is a high-level interface, implementing high-level methods such as [ICinemoPlayer::OpenTrack\(\)](#), [ICinemoPlayer::Play\(\)](#), [ICinemoPlayer::Stop\(\)](#), etc. All internal complexity such as graph building, codec selection, and so on, are hidden and inaccessible to calling applications. This ‘black-box’ approach is fundamental to Cinemo’s design goal: simply call [ICinemoPlayer::Play\(\)](#) and everything works. If some degree of internal configuration is required, it is always possible to achieve the required behaviour by means of [ICinemoConfig](#) configuration options.

Fundamental to Cinemo’s interface design is the concept that an application need not know what it is actually playing. This is the *lowest common denominator* approach. Until recently the lowest common denominator was DVD-video, which is now supplanted by blu-ray. The basic concept is, that if an application implements sufficient playback control and event handling to play a DVD, then it should also be able to play, for example, an MP3 file using exactly the same code. An MP3 file is simply regarded as a DVD with no menus, one title, no video angles and one audio language (note that with ID3 extensions, an MP3 file can have chapters).

This concept is fully implemented in [ICinemoPlayer](#). All supported media content can be played as if it were a DVD-video (or blu-ray disc). Playback control methods which are valid for DVD but not for the current media content are marked as *prohibited*, so the corresponding controls can be greyed out or hidden in an application’s user interface. The prohibited methods will additionally return an error code if called.

Note, this approach does not exclude an application from coding behaviour specific to the currently playing media content – for example, in the case of an MP3 or audio-only playback, an application may wish to display cover art, or some other graphical user interface, instead of blank video. An application can easily determine what is playing by means of [ICinemoPlayer](#) methods, or by monitoring playback events for the appearance and disappearance of video streams and ‘now playing’ metadata.

1.1.1 Virtual File System

Cinemo handles playback of media content from various media sources by means of a virtual file system. This implementation resides internally within Cinemo's 'black box' and is not directly exposed by any Cinemo SDK interface (although it is extensively used by all). The VFS is a layer of abstraction on top of more concrete file system implementations, such as those exposed by POSIX or UPnP Content Directory Service.

The VFS abstracts the following essential functions:

- Enumeration of folders or containers,
- Reading and writing of files, including live-streaming sources,
- Content change event notifications.

This layer of abstraction greatly simplifies the task of porting Cinemo to the wide variety of supported operating systems, extending support for new media sources, and coding navigators and metadata extraction, whose implementations never directly touch the underlying file system. Note that in some cases there may be a degree of overlap between the file systems supported by Cinemo and those natively supported on the target operating system. For example, Cinemo includes support for UDF and HFS file systems. In such cases it can be determined by the customer whether to use one or the other implementation.

Cinemo supports chaining of VFS objects. For example, it is possible to locate, play and extract metadata from an MP3 file inside a ZIP file located on the HFS partition on an ATAPI optical disc. In this example, three VFS objects are chained – ATAPI, HFS and ZIP – and the file contents will be dynamically ‘unzipped’ as required. Cinemo is able to play zipped DVD-video discs.

1.1.2 Navigators

Cinemo navigators are supplied media content by the VFS, and implement all playback control and event notification methods exposed by [ICinemoPlayer](#) interface. Navigators are additionally responsible for extracting metadata and de-multiplexing media content into their component audio, video and subpicture streams. A single navigator is typically implemented for each supported media format, but there is some overlap where a single navigator is responsible for multiple similar formats (e.g. MP4, AVI, 3GP).

It is the navigator's responsibility to implement the lowest common denominator approach to playback control, where all media formats are exposed as DVD-video (or blu-ray disc) compliant objects.

Cinemo navigators assist with content detection. Cinemo's approach is '*never trust the file extension*' and each navigator implementation will have the final say in whether a file is really an MP3, or as often occurs, an MP4 file masquerading as MP3. In the latter case, an alternative navigator will be automatically selected.

Cinemo navigators output 'now playing' metadata. This metadata, on start of playback, is extracted from media content headers and will be the same metadata as returned by [ICinemoVFS](#) during metadata extraction. However, metadata can change during playback – for example, DVD title change, SHOUTcast radio streaming protocol – and applications should be prepared to receive metadata update notifications at any time.

1.1.3 Decoders

Cinemo decoders are supplied elementary stream data by navigators, and are responsible for decoding and delivering uncompressed video frames or audio PCM to the renderers. Cinemo develops in-house a vast range of audio, video and subpicture decoders, either in software or utilizing available hardware resources, fully optimized for each supported target platform. A complete list of supported codecs can be obtained from your Cinemo representative.

1.1.4 Audio and Video Renderers

Cinemo renderers are the final stage in the playback pipeline, and are responsible for the synchronized output of decoded audio and video frames. Audio and video renderer implementations are platform specific, and are developed and optimized for each supported target platform.

1.1.5 Distributed Playback

Cinemo implements a proprietary Distributed Playback network protocol, where audio and video streams output from navigators are decoded and rendered locally, and also distributed to networked [ICinemoPlayer](#) instances for remote decoding and rendering. The [ICinemoPlayer](#) instance playing the media content is the master, and remote [ICinemoPlayer](#) instances receiving the streams are slaves. A proprietary clock synchronization protocol ensures that playback on master and slave are perfectly synchronized. Playback control methods on a slave are forwarded to the master for execution, enabling fully interactive control of playback on both master and slave.

1.2 Playback Example

The following sections, for the purpose of illustration, describe the steps required to perform playback of an MP3 file, with metadata extraction.

1.2.1 Configure Cinemo

Before obtaining any other Cinemo interface, it is required to obtain the [ICinemoConfig](#) interface and configure Cinemo Media Engine. There are many advanced configuration options here, but the default values are normally sufficient. However, it is always necessary to set the Cinemo plugin directory.

- [CinemoCreateConfig\(\)](#)
- [ICinemoConfig::SetPluginDirectory\(\)](#)

[ICinemoConfig::SetPluginDirectory\(\)](#) should only be called *once*. In the following sections various Cinemo API use cases are described, and in all cases the plug-in directory is set once at application start-up.

1.2.2 Create Playlist

Cinemo can only play tracks, and tracks can only exist in an [ICinemoPlaylist](#). So first it is necessary to create a playlist and, in this case, manually add the track.

- [CinemoCreatePlaylist\(\)](#)
- [ICinemoPlaylist::AppendTrack\(\)](#) with the URL of the MP3 file.

1.2.3 Create Player

Now we have a playlist, we can obtain its player interface.

- [ICinemoPlaylist::InitPlayback\(\)](#)

Now we have a player, we can play the track:

- [ICinemoPlayer::SetEventCallback\(\)](#)
- [ICinemoPlayer::SetTimeEventsInterval\(\)](#)
- [ICinemoPlayer::SetAudioParams\(\)](#)
- [ICinemoPlayer::SetVideoParams\(\)](#)
- [ICinemoPlayer::PlayTrack\(ICinemoPlaylist::GetTrack\(ICinemoPlaylist::GetCount\(\)\)\)](#)

1.2.4 Monitor Player Events

Since we specified an event callback, we will receive playback events.

- [CINEMO_EC_METADATA](#) – metadata is available.
- [CINEMO_EC_TIME](#) – elapsed time, refreshed every 250ms (configurable).
- [CINEMO_EC_FINISHED](#) – playback finished.

1.2.5 Extract Metadata

When we receive the [CINEMO_EC_METADATA](#) event, we know that new metadata has become available. To access the metadata, we need the [ICinemoMetapool](#) interface.

- [ICinemoPlayer::InitMetapool\(\)](#)

Now we have [ICinemoMetapool](#) interface, we can extract the metadata. Since there may be many kinds of metadata, including but not limited to track title, album, artist and cover art, we can loop through all items in the pool and extract the metadata we need, or search for a specific metadata item.

- [ICinemoMetapool::GetItemCount\(\)](#)
- [ICinemoMetapool::GetItem\(\)](#) or [ICinemoMetapool::Find\(\)](#)
- [ICinemoMetapool::Read\(\)](#)
- [ICinemoMetapool::GetText\(\)](#) or [ICinemoMetapool::GetBlob\(\)](#)

1.3 Playback of Blu-ray Discs

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

To support playback of blu-ray discs, a new [ICinemoPlayer2](#) interface has been introduced. This new interface also supports playback of all the other media formats currently supported by [ICinemoPlayer](#), including DVD and media files, but [ICinemoPlayer](#) cannot play blu-ray discs.

The following sections describe the differences how Blu-ray capable playback should be implemented compared to legacy media playback.

1.3.1 Interaction Model

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Blu-ray gives most of the interaction control to the content applications which are loaded from the disc and are executed in order to control the AV playback. Thus the Cinemo Blu-ray navigator represents an execution environment for Blu-ray content applications. This differs from DVD where the navigator controls the AV playback of the DVD. Simplified it could be stated that every Blu-ray disc comes with its own navigator.

Key events should generate both pressed and release events. To fully support this interaction model the [ICinemoPlayer2](#) interface should be used instead of [ICinemoPlayer](#). It will allow passing user key and mouse events to the player.

When support for Blu-ray is needed the [ICinemoPlayer2](#) interface should be used. The key event based user interaction can also be used with any other media type. There is no need to implement two versions of the player for Blu-ray and non-Blu-ray content.

The [ICinemoPlayer2](#) interface does not expose the following functions; instead user key and mouse events should be posted:

- [ICinemoPlayer::SetSpeed\(\)](#)
- [ICinemoPlayer::Stop\(\)](#)
- [ICinemoPlayer::NextChapter\(\)](#)
- [ICinemoPlayer::PreviousChapter\(\)](#)
- [ICinemoPlayer::SelectButton\(\)](#)
- [ICinemoPlayer::ActionButton\(\)](#)
- [ICinemoPlayer::SelectButtonRelative\(\)](#)
- [ICinemoPlayer::SelectButtonPosition\(\)](#)
- [ICinemoPlayer::ActionButtonPosition\(\)](#)

Some features are implemented for other media but are not available for blu-ray discs:

- [ICinemoPlayer2::SetScanTime\(\)](#)
- [ICinemoPlayer2::ShowMenu\(\)](#)
- [ICinemoPlayer2::ReturnFromSubmenu\(\)](#)

- [ICinemoPlayer2::ReplayChapter\(\)](#)

The following features are only available with certain types of media (e.g. DVD, AVCHD):

- [ICinemoPlayer2::SaveState\(\)](#)
- [ICinemoPlayer2::RestoreState\(\)](#)

1.3.2 Fully Asynchronous Model

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

With other media formats Cinemo Media Engine did process some of the commands synchronously and return error values when the action failed. This is not the case when using the [ICinemoPlayer2](#) interface. User operations and key event handling is completely asynchronous. Feedback can be checked via monitoring the player events.

1.3.3 Disc Ejection

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Blu-ray content application needs to know about disc ejecting before the disc is actually ejected. Terminating playback without proper closing of the content application can leave the persistent storage in an invalid state. This can cause problems for later playback if the content application is poorly implemented.

Before ejecting the disc the application should always call [ICinemoPlayer2::CloseTrack\(\)](#) and wait for the `CINEMO_EC_CLOSE` event. Releasing the [ICinemoPlayer2](#) object would also work, but it would block the calling thread until the content application is shutdown.

After closing the Blu-ray track it's possible the player status doesn't return to `CINEMO_STATUS_NONE` status but stays in `CINEMO_STATUS_WAIT_DISC`. In this case the BD-J application is still running in the background and expecting the next disc to be inserted. Once the disc insertion is detected [ICinemoPlayer2::PlayTrack\(\)](#) should be called again to resume playback.

1.4 UPnP Discovery

The following sections describe how to use Cinemo API to discover UPnP media servers, or other UPnP devices, on the network. If you already know the IP address, network path and attributes of the device you wish to browse, skip to the next section.

Media servers and other UPnP devices advertise themselves on a network by means of SSDP (Simple Services Discovery Protocol). Cinemo API exposes SSDP as a virtual file system, which supports enumeration of UPnP devices on the network by means of SSDP in the same way as, for example, files in a folder, or CD tracks on a CD.

UPnP devices may appear and disappear from the network at any time, so UPnP discovery is a continuous process. Cinemo API provides low-level blocking methods, and because these methods are blocking, the application should create one or more background threads dedicated to these functions. It is the application's responsibility to manage thread state and synchronization.

1.4.1 Configure Cinemo

Before obtaining any Cinemo interface, it is required to obtain the [ICinemoConfig](#) interface and configure Cinemo Media Engine. Note, [ICinemoConfig::SetPluginDirectory\(\)](#) should be called *once*. If you have already performed this step (e.g. at application start-up), then skip this section.

- [CinemoCreateConfig\(\)](#)
- [ICinemoConfig::SetPluginDirectory\(\)](#)

1.4.2 Open SSDP

Cinemo provides a [ICinemoVFS](#) interface which can be used to browse or search virtual file systems. SSDP is supported by this interface and exposed as a virtual file system. All [ICinemoVFS](#) objects are specified by URL. In this case, the URL must include the standard multicast address for SSDP, with an additional parameter to specify the URN of the device or service we are interested in discovering.

```
CinemoAutoPtr<ICinemoVFS> pvfs;
CinemoAutoPtr<ICinemoMetapool> ppool;
CinemoError hr;
CinemoVFSAttributes attrs;

const char * szurl =
    "ssdp://239.255.255.250:1900?"
    "urn=urn:schemas-upnp-org:device:MediaServer:1";

if (hr = CinemoCreateVFS(pvfs.pp())) {
    return hr;
} else if (hr = pvfs->Open(szurl, CINEMO_VFS_OPEN_FOLDER, attrs, ppool.pp())) {
    return hr;
}
```

The [ICinemoVFS::Open\(\)](#) method returns [ICinemoMetapool](#) which describes the known content of the virtual file system in its entirety. All discovered UPnP devices are represented as metadata items with title attribute

advancing from 1 to the number of devices. The following example displays the content of [ICinemoMetapool](#) where a single UPnP media server device was discovered on the network:

```
opening...
path: ssdp://239.255.255.250:1900?urn=urn:schemas-upnp-org:device:MediaServer:1
type: FOLDER | VIRTUAL | METADATA_COMPLETE | WATCHABLE | SSDP

Title 01:
VFS_PATH: upnp://127.0.0.1:49158/0/MediaServer/DeviceDesc.xml
VFS_NAME: Cinemo Media Server
VFS_ICON: 12179 bytes
VFS_TYPE: UPNP_DEVICE
VFS_SSDP_URN: urn:schemas-upnp-org:device:MediaServer:1
VFS_SSDP_LOCATION: http://127.0.0.1:49158/0/MediaServer/DeviceDesc.xml
VFS_SSDP_USN: uuid:12345678-0000-0000-0000-00007f9f3aa6::urn:schemas-upnp-org:device
VFS_SSDP_SERVER: Cinemo 1.5.0.12224
VFS_UPNP_CTRL: /ctl/ContentDir
VFS_UPNP_SUBSCRIBE: /evt/ContentDir
VFS_UPNP_PLATFORM: vs_2008
```

The calling application should examine the content of the returned [ICinemoMetapool](#) and extract the required information. In the above example, the media server name, icon and location are the most important items, and may be used to populate e.g. a list control in the application user interface. [CINEMO_METANAME_VFS_SSDP_USN](#) is the unique serial number of the UPnP device.

1.4.3 Watch Changes

The [ICinemoVFS::Open\(\)](#) method returned a [CinemoVFSAttributes](#) structure which describes the attributes of the VFS. In this case, the *type* bitmask of attributes includes the following flags:

- [CINEMO_VFS_FLAG_FOLDER](#)
- [CINEMO_VFS_FLAG_VIRTUAL](#)
- [CINEMO_VFS_FLAG_METADATA_COMPLETE](#)
- [CINEMO_VFS_FLAG_WATCHABLE](#)
- [CINEMO_VFS_PROTOCOL_SSDP](#)

[CINEMO_VFS_FLAG_WATCHABLE](#) indicates that [ICinemoVFS](#) may be watched for changes. This is required for SSDP because devices can appear and disappear from the network at any time. To watch for changes, call:

- [ICinemoVFS::Watch\(\)](#)

This method will return a new [ICinemoMetapool](#) interface whenever a change is detected, i.e. when UPnP devices appear or disappear from the network. In the same way as for [ICinemoVFS::Open\(\)](#), the application should examine the content of the returned [ICinemoMetapool](#) and extract the required information.

1.4.4 Cancel/Enable the VFS

Both [ICinemoVFS::Open\(\)](#) and [ICinemoVFS::Watch\(\)](#) are blocking methods. They will not return until new metadata is actually available. Because these methods are blocking, the application is responsible for creating

worker threads to call these methods. If an application wishes to cancel a call to [ICinemoVFS::Open\(\)](#) or [ICinemoVFS::Watch\(\)](#) – for example, because it wishes to exit – then the blocking methods may be manually unblocked:

- [ICinemoVFS::Cancel\(\)](#)
- [ICinemoVFS::Enable\(\)](#)

[ICinemoVFS::Cancel\(\)](#) will cause the current and all future calls to [ICinemoVFS::Open\(\)](#) or [ICinemoVFS::Watch\(\)](#) to return immediately with [CinemoErrorCancel](#). The cancel operation is sticky: this means the [ICinemoVFS](#) interface will remain in cancelled state until [ICinemoVFS::Enable\(\)](#) is called. The cancel operation is reference counted, which means, to re-enable [ICinemoVFS](#) there must be as many calls to [ICinemoVFS::Enable\(\)](#) as there were to [ICinemoVFS::Cancel\(\)](#).

1.5 Browse UPnP Media Servers

The following sections describe how to use Cinemo API to browse the contents of a UPnP media server on the network. Before using these interfaces, the URL of the media server must be known. If it is not known, check the previous sections for [UPnP Discovery](#).

The UPnP browsing example given here makes use of the [ICinemoPlaylist](#) interface. This is because, in addition to browsing, we often want to play the results. [ICinemoPlaylist](#) also implements a fetch-on-demand strategy which makes it much more efficient to browse media servers with large numbers of items.

[ICinemoPlaylist](#) is designed to support browsing of UPnP media servers with very large numbers of items, e.g. in excess of 1,000,000 tracks. Since this number of items should never be fetched from a media server in a single operation, [ICinemoPlaylist](#) supports dynamic fetching of items on demand. Fetching of items in a playlist may be triggered by the user interface, e.g. when it wishes to display the attributes of a track in a list control, or by attached [ICinemoPlayer](#) interfaces, which may wish to play different tracks to those currently displayed by the user interface. This is managed for you by [ICinemoPlaylist](#). From the application point of view, browsing and playing the contents of a remote UPnP media server is no different to browsing and playing local folders and files.

1.5.1 Configure Cinemo

Before obtaining any Cinemo interface, it is required to obtain the [ICinemoConfig](#) interface and configure Cinemo Media Engine. Note, [ICinemoConfig::SetPluginDirectory\(\)](#) should only be called *once*. If you have already performed this step (e.g. at application start-up), then skip this section.

- [CinemoCreateConfig\(\)](#)
- [ICinemoConfig::SetPluginDirectory\(\)](#)

1.5.2 Create Playlist

We will use a playlist to represent the contents of the UPnP media server, but instead of manually adding tracks, we will use the [ICinemoPlaylist::Open\(\)](#) method:

- [CinemoCreatePlaylist\(\)](#)
- [ICinemoPlaylist::Open\(\)](#) with URL of the UPnP media server.

The URL of the media server may already be known, because the media server was created with a known IP address and port number, or it may be obtained by [UPnP Discovery](#), in which case, the URL is specified by [CINEMO_METANAME_VFS_PATH](#) metadata returned from SSDP. In all cases, when browsing media servers, the URL must include the following URL parameters:

- **pid** – the parent container ID we wish to browse or search. The ID of the root container of all media servers is always 0 (zero), in accordance with UPnP specification. Other ID's can be obtained by first browsing the root container of the media server and extracting the [CINEMO_METANAME_VFS_UPNP_ID](#) metadata of the resulting children (which must themselves be indicated as containers by examining the [CINEMO_METANAME_VFS_TYPE](#) attribute).

- **browse** or **search** – the UPnP browse or search command.

The application is responsible for appending the required URL parameters. Note, the application should take care that the base URL of a media server may already contain URL parameters – which is often the case, for example with Microsoft UPnP servers. In this case, **pid** and **browse** should be specified as *additional* URL parameters.

Some example URLs are given below:

```
upnp://127.0.0.1:49158/0/MediaServer/DeviceDesc.xml?
pid=0&
browse=BrowseDirectChildren
```

This URL will open the root container, and browse the immediate children. Note that the first portion of the URL is identical to the URL obtained from [CINEMO_METANAME_VFS_PATH](#) in the previous section on [UPnP Discovery](#).

```
upnp://127.0.0.1:49158/0/MediaServer/DeviceDesc.xml?
pid=0&
browse=BrowseMetadata
```

This URL will open the root container, and instead of browsing children, will return the container itself, along with its associated metadata. The resulting [ICinemoPlaylist](#) will contain only a single track.

```
upnp://127.0.0.1:49158/0/MediaServer/DeviceDesc.xml?
pid=0&
search=(upnp:class derivedfrom "object.item" and (
dc:title contains "madonna" or
upnp:album contains "madonna" or
upnp:artist contains "madonna" or
upnp:albumArtist contains "madonna") and (@refID exists false))
```

This URL will open the root container, and search for all items (not containers, and not links to items) which contain the text *madonna*. This is an example of a complex UPnP search command.

1.5.3 Displaying Items in a User Interface

When [ICinemoPlaylist::Open\(\)](#) returns success, the playlist is created. The number of tracks in the playlist will be known, and the track identifiers of each track will be known. However, due to the fetch-on-demand strategy employed by [ICinemoPlaylist](#), the metadata for each track – including the playback URL – may not yet be fetched from the server. For displaying tracks in a user interface, the following methods can be used:

- [ICinemoPlaylist::GetURL\(\)](#)
- [ICinemoPlaylist::GetMetapool\(\)](#)

Both methods take an additional flags argument, which specify if the operation should be executed synchronously or asynchronously, and if unfetched metadata should be fetched from the server.

For synchronous operation, `CINEMO_PLFLAGS_READ | CINEMO_PLFLAGS_WAIT` should be specified. In this case, if the URL or metadata for the given track has not yet been fetched from the media server, the method will block until the track has been fetched. If the track has already been fetched, the method will return immediately with the corresponding [ICinemoUTF8](#) or [ICinemoMetapool](#) interface.

For asynchronous operation, `CINEMO_PLFLAGS_READ` should be specified. In this case, if the URL or metadata for the given track has not yet been fetched from the media server, the metadata will be queued for fetching, and the method will return immediately with [CinemoErrorPending](#). In this case, [ICinemoPlaylist](#) will post [EC_PLAYLIST_CHANGED](#) when the metadata has been fetched from the server.

If neither `CINEMO_PLFLAGS_READ` or `CINEMO_PLFLAGS_WAIT` are specified, these methods can be used to determine if metadata is available for a given track, without requesting it to be fetched.

1.5.4 Playing Tracks

Playing UPnP tracks is the same as playing any other kind of tracks in [ICinemoPlaylist](#). First, the [ICinemoPlayer](#) interface must be obtained:

- [ICinemoPlaylist::InitPlayback\(\)](#)

Any track in the playlist can then be played with the returned [ICinemoPlayer](#) interface. [ICinemoPlaylist](#) will automatically fetch the required metadata for playback – i.e. the playback URL – on demand from the media server when required to do so by [ICinemoPlayer](#). This occurs automatically in the background.

1.6 USB Devices

Cinemo Media Engine supports several different USB device types, like iAP, MTP or MSD. While the device discovery process needs to be handled outside of Cinemo, the supported USB communication protocols can be detected using Cinemo. This is described in the section [USB Detection](#).

USB device detection, browsing and playback functions are accessed with standard Cinemo interfaces, e.g. [ICinemoVFS](#), [ICinemoPlaylist](#) and [ICinemoPlayer](#). This is in accordance with Cinemo API design goals. From an API perspective, there is no difference between browsing and playback of an MTP or iAP device, a UPnP media server, a CD or a folder on your hard disk. The same application code may be used to support all of these use-cases.

1.6.1 USB Discovery

Do not use this interface for production grade software!

For internal testing and demo showcases Cinemo Media Engine provides USB discovery tools. However, these tools are not designed to be used in production grade software, as they contain several deficiencies.

The [ICinemoVFS](#) interface is used for USB discovery, in the same way as used for [UPnP Discovery](#) described in previous sections. Before starting, an application should ensure that Cinemo has been correctly configured using the [ICinemoConfig](#) interface.

To search for all USB devices, call [ICinemoVFS::Open\(\)](#) with the following path:

- [ICinemoVFS::Open\("usb://"\)](#)

To search for USB devices with a specific vendor ID, the vendor ID may be added to the path as a URL parameter. For instance, the vendor ID 1452 (0x05ac) is for Apple devices:

- [ICinemoVFS::Open\("usb://?vid=1452"\)](#)

The [ICinemoVFS::Open\(\)](#) method returns an [ICinemoMetapool](#) interface to the metadata for each discovered USB device. Each device is represented as metadata items with a title attribute range from one to the total number of discovered devices. The following example shows the metadata content with one Apple device and one MTP device:

```

title 01:
VFS_PATH: "iap://usb:///dev/bus/usb/003/009"
VFS_NAME: "iPhone"
VFS_TYPE: FOLDER | DEVICE | VIRTUAL | ORDERED | METADATA_COMPLETE | WATCHABLE | USB | IAP
VFS_USB_ID: 512
VFS_USB_DEVICE_CLASS: 0
VFS_USB_DEVICE_SUBCLASS: 0
VFS_USB_DEVICE_PROTOCOL: 0
VFS_USB_MAX_PACKET_SIZE: 64
VFS_USB_VENDOR_ID: 1452
VFS_USB_PRODUCT_ID: 4776
VFS_USB_DEVICE_ID: 1794
VFS_USB_MANUFACTURER: "Apple Inc."

```

```
VFS_USB_PRODUCT: "iPhone"
VFS_USB_SERIAL_NO: "e9af9a6db5ad80727b81e1403ee58aef82c7141"
VFS_USB_NUM_CONFIGURATIONS: 4
VFS_USB_CARPLAY_SUPPORTED: 1

title 02:
VFS_PATH: "mtp://usb:///dev/bus/usb/003/011"
VFS_NAME: "Nexus 5X"
VFS_TYPE: FOLDER | DEVICE | VIRTUAL | ORDERED | METADATA_COMPLETE | WATCHABLE | USB | MTP
VFS_USB_ID: 512
VFS_USB_DEVICE_CLASS: 0
VFS_USB_DEVICE_SUBCLASS: 0
VFS_USB_DEVICE_PROTOCOL: 0
VFS_USB_MAX_PACKET_SIZE: 64
VFS_USB_VENDOR_ID: 6353
VFS_USB_PRODUCT_ID: 20194
VFS_USB_DEVICE_ID: 784
VFS_USB_MANUFACTURER: "LGE"
VFS_USB_PRODUCT: "Nexus 5X"
VFS_USB_SERIAL_NO: "005115aa106fcd63"
VFS_USB_NUM_CONFIGURATIONS: 1
```

Since USB discovery is a dynamic process, an application should call [ICinemoVFS::Watch\(\)](#) after calling [ICinemoVFS::Open\(\)](#). The [ICinemoVFS::Watch\(\)](#) method is blocking, so depending on application requirements, it is expected to be called on a background thread. The [ICinemoVFS::Watch\(\)](#) method, like [ICinemoVFS::Open\(\)](#), will return a new [ICinemoMetapool](#) containing a complete snapshot of the USB discovery state, whenever the state changes (because a device was inserted and removed). It is the calling application's responsibility to process the resulting metadata and calculate the *delta*, in order to determine which devices, if any, were inserted or removed.

It is important to note that the metadata returned from USB discovery is limited to information which can be extracted from USB device descriptors. The USB device has not yet been opened.

The USB discovery process detects the type of the device – for example, MTP or iAP – based on information it finds in USB device descriptors, and returns this information encoded in [CINEMO_METANAME_VFS_TYPE](#). However, it does not detect which concrete protocol – for example, iAP1 or iAP2 – the device supports. The protocol can only be determined by opening the device using the [CINEMO_METANAME_VFS_PATH](#). Likewise, in the above example, the [CINEMO_METANAME_VFS_NAME](#) attribute has been derived from information found in USB device descriptors, such as the USB vendor and product ID. This may not be the actual name of the USB device – for example, “iPhone”. In order to get more detailed and accurate information, each device must be individually opened using the [CINEMO_METANAME_VFS_PATH](#) attribute.

1.6.2 USB Detection

USB detection can be used to determine the type of a USB devices independently of USB discovery. Currently detection of MTP, iAP and Android-Auto devices is supported. To determine the type of a USB device, call [ICinemoVFS::Open\(\)](#) with the platform specific USB device <identifier>.

- [ICinemoVFS::Open\("usbdetect://usb://<identifier>"\)](#)

For example, on a Linux system, the USB device <identifier> will be the POSIX mounted path of the USB device, and the URL will therefore be similar to "usbdetect://usb:///dev/bus/usb/001/035".

The [ICinemoVFS::Open\(\)](#) method returns an [ICinemoMetapool](#) interface to the metadata for the detected USB device. It also returns a [CinemoVFSAttributes](#) structure which describes the attributes of the VFS. The *type* bitmask of attributes may include the following flags:

VFS_TYPE	USB Device Type
CINEMO_VFS_SUBTYPE_MTP	MTP
CINEMO_VFS_SUBTYPE_IAP	iAP
CINEMO_VFS_SUBTYPE_AOAP	Android Open Accessory Protocol
CINEMO_VFS_SUBTYPE_AOAP_READY	Android Open Accessory Protocol capable
CINEMO_VFS_SUBTYPE_ADB	Android debug bridge

If one USB device supports more than one protocol, the returned [ICinemoMetapool](#) contains all the instances that can co-exist. For instance, the following device supports both MTP and AOAP (e.g. Android Auto):

```

title 01:
VFS_PATH: "mtp://usb:///dev/bus/usb/003/011"
VFS_NAME: "Nexus 5X"
VFS_TYPE: FOLDER | DEVICE | VIRTUAL | ORDERED | METADATA_COMPLETE | WATCHABLE | USB | MTP
VFS_USB_ID: 512
VFS_USB_DEVICE_CLASS: 0
VFS_USB_DEVICE_SUBCLASS: 0
VFS_USB_DEVICE_PROTOCOL: 0
VFS_USB_MAX_PACKET_SIZE: 64
VFS_USB_VENDOR_ID: 6353
VFS_USB_PRODUCT_ID: 20194
VFS_USB_DEVICE_ID: 784
VFS_USB_MANUFACTURER: "LGE"
VFS_USB_PRODUCT: "Nexus 5X"
VFS_USB_SERIAL_NO: "005115aa106fcd63"
VFS_USB_NUM_CONFIGURATIONS: 1

title 02:
VFS_PATH: "aoap://usb:///dev/bus/usb/003/011"
VFS_NAME: "Nexus 5X"
VFS_TYPE: FOLDER | DEVICE | VIRTUAL | ORDERED | METADATA_COMPLETE | WATCHABLE | USB |
AOAP_READY
VFS_USB_ID: 512
VFS_USB_DEVICE_CLASS: 0
VFS_USB_DEVICE_SUBCLASS: 0
VFS_USB_DEVICE_PROTOCOL: 0
VFS_USB_MAX_PACKET_SIZE: 64
VFS_USB_VENDOR_ID: 6353
VFS_USB_PRODUCT_ID: 20194
VFS_USB_DEVICE_ID: 784
VFS_USB_MANUFACTURER: "LGE"

```

```
VFS_USB_PRODUCT: "Nexus 5X"
VFS_USB_SERIAL_NO: "005115aa106fcd63"
VFS_USB_NUM_CONFIGURATIONS: 1
```

The protocol “usbdetect://” should be used only for device detection. For PTP devices the VFS_USB_PTP will be set to 1.

1.6.3 USB Device Lifecycle

When opening a new USB device it is mandatory to create a so-called lifecycle object, which will represent the physical connection to the device. As soon as this lifecycle object is closed or released, Cinemo will immediately sever all connections to the device and release all system resources associated with this connection, e.g. USB endpoints.

A lifecycle object may be a simple [ICinemoVFS](#) interface, which has been opened with the `CINEMO_VFS_OPEN_DEVICE` flag.

If a device is opened without using a lifecycle object, all Cinemo interfaces will return with `CinemoErrorDeviceLifecycle`. This indicates that either there is no lifecycle object yet and no device flag has been given or that there is already a lifecycle object and also the device flag has been given.

1.6.4 Browsing and Playback

USB devices expose a native track hierarchy which may be browsed using [ICinemoPlaylist](#). The root hierarchy may be opened with [ICinemoPlaylist::Open\(\)](#), using the `CINEMO_METANAME_VFS_PATH` attribute obtained from the USB discovery process. For example:

- `ICinemoPlaylist::Open("usbdetect://usb://\\.\libusb0-0002-0x05ac-0x12a0")`

This method will internally execute the following functions:

1. Open the device,
2. Select a communication protocol (MTP, iAP-1, or iAP-2),
3. Perform authentication (iAP only),
4. Populate the [ICinemoPlaylist](#) with the root hierarchy of the USB device.

When browsing USB devices, the resulting [ICinemoPlaylist](#) may contain both folders and tracks. The folders may be recursively browsed with additional calls to [ICinemoPlaylist::Open\(\)](#), and the tracks may be played with [ICinemoPlayer::PlayTrack\(\)](#), in the same way as for any other [ICinemoPlaylist](#).

To recursively browse into a folder of the native device hierarchy, it is necessary to obtain the [ICinemoMetapool](#) interface for the folder item by means of [ICinemoPlaylist::GetMetapool\(\)](#), read the `CINEMO_METANAME_VFS_TYPE` attribute to determine that the item is actually a browsable folder, and call [ICinemoPlaylist::Open\(\)](#) with `CINEMO_METANAME_VFS_PATH`.

In the same way as for UPnP playlists, [ICinemoPlaylist](#) implements a fetch-on-demand strategy designed to support browsing of USB devices with very large numbers of tracks. The fetching of tracks may be triggered by [ICinemoPlaylist](#) operations such as [ICinemoPlaylist::GetMetapool\(\)](#) or [ICinemoPlaylist::GetURL\(\)](#),

or internally by [ICinemoPlayer](#) operations such as [ICinemoPlayer::PlayTrack\(\)](#). This is managed transparently by the [ICinemoPlaylist](#) implementation.

1.6.5 MTP

The implementation of MTP is based upon the specification "Media Transfer Protocol, Revision 1.1". Cinemo supports the following URIs for MTP:

1. *mtp://* identifies the containers for devices, storages and generic folders;
2. *mtpfile://* identifies the "objects" that are not of association-types;
3. *mtpplaylist://* identifies the abstract associations (e.g., "Abstract Multimedia Album");

User application is able to access MTP devices via the standard Cinemo interfaces (e.g.[ICinemoVFS](#), [ICinemoPlaylist](#), [ICinemoPlayer](#)) for playback and browsing. Cinemo opens MTP session once user application creates the lifecycle object for the MTP device. The session will be closed if the lifecycle object is released. Cinemo will fire [CINEMO_EC_ERROR](#) to the watcher of the VFS instance of device, if the following errors are detected in MTP sessions:

1. [CinemoErrorNotConnected](#)
2. [CinemoErrorTimeout](#)
3. [CinemoErrorBusy](#)
4. [CinemoErrorMTPProtocol](#)
5. [CinemoErrorMTPSessionNotOpen](#)

User application can re-open the MTP session upon the above errors except [CinemoErrorNotConnected](#) by re-creating the lifecycle object.

Cinemo has some special code to handle Samsung devices. To suppress the warning dialog on a Samsung device when connecting it, cinemo needs to send a persistent UUID to the device. If the UUID is acknowledged by the Samsung device, it will be stored on the device and the warning dialog will not appear any more when the same UUID is used in the subsequent connections. The option [CINEMO_OPTION_MTP_UUID](#) is intended for user applications to specify the desired UUID. It is responsibility for the user application to store the option. If the option has the special value "_seed_" which is also the default value, cinemo will generate a UUID by seeding some constant system-settings. The seeded UUIDs might not be globally unique. If the option is empty, no UUID will be sent to Samsung devices.

Devices-List

With the option [CINEMO_OPTION_MTP_DEVICELIST](#) user applications are able to customize the behavior of the MTP module for specific devices. The option contains an XML file. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<devices>
    <!-- some special MTP device -->
    <device vid="12AB" pid="30E1" enable="true">
        <usb cnfg="2" intfd="1" alt="0" />
        <timeouts os="5000" goh="10000" other="2000" />
    
```

```

        <flags>GETOBJECTPROPLIST_BROKEN|OPENSESSION_AT_FIRST</flags>
    </device>
</devices>
```

Cinemo maintains an internal hardcoded device-list. The internal device-list is well-tested and can be overwritten by the option. The schema of the XML is described in the following XSD and it can also be used to validate the XML.

```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="devices">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="device" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <!-- This element specifies explicitly which USB interface is MTP. If absent, the
                            interface will be determined automatically.-->
                        <xs:element name="usb" minOccurs="0" maxOccurs="1">
                            <xs:complexType>
                                <!-- bConfigurationValue -->
                                <xs:attribute name="cnfg" type="xs:integer" />
                                <!-- bInterfaceNumber -->
                                <xs:attribute name="intfd" type="xs:integer"/>
                                <!-- bAlternateSetting -->
                                <xs:attribute name="alt" type="xs:integer"/>
                            </xs:complexType>
                        </xs:element>
                        <!-- This element sets the expiration-time in milliseconds for specific MTP
                            operations -->
                        <xs:element name="timeouts" minOccurs="0" maxOccurs="1">
                            <xs:complexType>
                                <!--the timeout for OpenSession -->
                                <xs:attribute name="os" type="xs:integer"/>
                                <!--the timeout for GetObjectHandles -->
                                <xs:attribute name="goh" type="xs:integer"/>
                                <!--the timeout for other MTP operations -->
                                <xs:attribute name="other" type="xs:integer"/>
                            </xs:complexType>
                        </xs:element>
                        <!-- This element is used to customize the behavior for the device -->
                        <!-- OPENSESSION_AT_FIRST: the device requires the "OpenSession" as the first
                            operation; -->
                        <!-- GETOBJECTPROPLIST_BROKEN: avoid using the operation "GetObjectPropList"; -->
                        <!-- GETOBJECTPROPSSUPPORTED_BROKEN: avoid using the operation "
                            GetObjectPropsSupported"; -->
                        <!-- DEVICEFRIENDLYNAME_BROKEN: avoid reading the property "DeviceFriendlyName"; -->
                        <!-- GETNUMOBJECTS_BROKEN: avoid calling "GetNumObjects"; -->
                        <!-- PTP: the device is a PTP explicitly -->
                        <!-- NO_PTP: the device is not a PTP explicitly -->
                    <xs:element name="flags" minOccurs="0" maxOccurs="1">
                </xs:sequence>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```

<xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:pattern value="_a_pipe_separated_list_of_the_flags_"/>
    </xs:restriction>
</xs:simpleType>
</xs:element>
</xs:sequence>
<!-- vendor id (hex) -->
<xs:attribute name="pid" type="U16BitHexInt" use="required" />
<!-- product id (hex) -->
<xs:attribute name="vid" type="U16BitHexInt" use="required" />
<!-- the value 'false' indicates to block the device and all other settings will be
ignored -->
<xs:attribute name="enable" type="xs:boolean" default="true"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:simpleType name="U16BitHexInt">
    <xs:union memberTypes="xs:unsignedInt">
        <xs:simpleType>
            <xs:restriction base="xs:token">
                <xs:pattern value="[0-9A-Fa-f]{4}"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
</xs:simpleType>
</xs:schema>

```

Native metadata

The "native metadata" means the "object-properties" defined in MTP specification. By default, Cinemo scans the metadata from media-files on MTP devices by itself. In other words, Cinemo does not read the object-properties provided by MTP devices. If a MTP device supports the advanced MTP operations like "GetObjectPropList" (0x9805), Cinemo will be able to retrieve the native metadata from the device.

To enable the feature, please set *CINEMO_OPTION_MTP_NATIVEMETADATA* to *true*. If Cinemo retrieves the native metadata successfully when browsing a folder, the *CINEMO_VFS_FLAG_METADATA_COMPLETE* will be set to each child. Even if the option is enabled, Cinemo might fail to obtain the object-properties for different reasons and the *FLAG_METADATA_COMPLETE* will not be set in that case. In all the cases, user applications can force Cinemo to scan the metadata of a track by opening the VFS of the track. By the way, on some Samsung devices the browsing of folders will be slowed down due to some known issues, if the option is enabled.

The mapping from "object-properties" defined in MTP specification to Cinemo's metaname is listed in the following table.

PropertyCode	PropertyName	Metaname
0xDC07	ObjectFileName	CINEMO_METANAME_VFS_NAME
0xDC09	DateModified	CINEMO_METANAME_VFS_TIME
0xDC41	PersistentUniqueObjectIdentifier	CINEMO_METANAME_VFS_UUID
0xDC46	Artist	CINEMO_METANAME_ARTIST
0xDC89	Duration	CINEMO_METANAME_DURATION
0xDC96	Composer	CINEMO_METANAME_COMPOSER
0xDC99	OriginalReleaseDate	CINEMO_METANAME_DATE
0xDC9A	AlbumName	CINEMO_METANAME_ALBUM
0xDC9B	AlbumArtist	CINEMO_METANAME_ALBUM_ARTIST
0xDC04	ObjectSize	CINEMO_METANAME_VFS_SIZE

1.7 Apple Devices

Cinemo Media Engine supports discovery, browsing, indexing and playback of Apple devices. Both iAP-1 and iAP-2 protocols are supported. USB device mode is supported on all platforms. USB host mode, which has dependencies on driver and system hardware configuration, can be enabled on a project specific basis.

For iAP-1, the following lingos are used:

- 0x00 General Lingo
- 0x03 Display Remote Lingo
- 0x04 Extended Interface Lingo
- 0x0a Digital Audio Lingo

With custom identification the other lingos specified by Apple can be utilized too.

For iAP-2, the following session types are supported:

- 0x00 Control Session
- 0x01 File Transfer Session
- 0x02 External Accessory Session

Like for other USB devices, iAP devices also require a [lifecycle object](#). This may be an [ICinemoVFS](#) interface or the initial [ICinemolAP](#) instance, which can be given the `CINEMO_IAP_OPEN_DEVICE` flag.

Browsing and playback of Apple devices can be realized with [ICinemoVFS](#), [ICinemoPlaylist](#) and [ICinemoPlayer2](#) interfaces, just like for any other media source.

Apple devices protocols additionally support non-multimedia functions which may be required by a calling application. Since the connection to an Apple device, unlike mass storage devices, cannot be shared by multiple processes, Cinemo provides instances of the [ICinemolAP](#) interface, which act as a proxy to the single iAP connection and can be instantiated several times, thus supporting parallel access from within one process.

In combination with Cinemos DDP feature it is possible to share a single connection to an Apple device from within multiple processes.

1.7.1 USB Host Mode

Apple devices may be capable of taking both USB host and USB device roles when connected to an accessory. USB host mode refers to the case where the Apple device is the USB host, and the accessory is the USB device. Apple specification indicates that USB host mode is *recommended* for all new accessory designs.

Cinemo Media Engine is designed to transparently handle the connection with an Apple device independently of the USB mode. All device communication is performed via the [Cinemo Transport Library](#), which can be implemented with the provided interface.

A custom transport library implementation may use BSP specific device drivers to access Apple devices in both USB device and host mode, connected via Bluetooth or WiFi.

1.7.2 Authentication

In order to work with Apple devices, an Apple Authentication Coprocessor must be accessible on the target system. Cinemo supports coprocessor versions 2.0B and 2.0C. The application must configure the authentication URL such that Cinemo can communicate with the Apple Authentication Coprocessor.

When accessing the Apple Authentication Coprocessor for different applications like CarPlay and iAP connections, the access to the chip needs to be synchronized. Cinemo Media Engine provides the [ICinemoAppleAuth](#) interface which may be used to access the chip in a thread-safe way from different processes.

Additionally, when implementing a [Cinemo Apple Authentication Library](#), it is possible to access an Apple Authentication Coprocessor using customized BSP drivers.

1.7.3 Browsing

Apple device browsing and playback functions are accessed using standard Cinemo interfaces. For example, the root hierarchy of an Apple device can be opened for browsing with [ICinemoPlaylist](#) with the following call, where the Apple device path has been obtained through [USB Discovery](#):

- `ICinemoPlaylist::Open("iap://usb:///dev/bus/001/001")`

The resulting playlist, after opening the root hierarchy of an Apple device, will contain the following tracks:

1. The '*Now Playing*' track
2. The '*Now Playing list*' (for iAP-2 restrictions explained below might apply)
3. The '*Device Native Browse Hierarchy*' container (for iAP-2 restrictions explained below might apply)

The '*Now Playing*' track may be played by [ICinemoPlayer2](#), which will render any audio stream and metadata currently played on the Apple device. The name of the '*Now Playing*' track is the title of the currently playing track. Playback of the '*Now Playing*' track will not cause any track change or other control command to be issued to the device, so if the device is currently not playing any tracks, playback of the '*Now Playing*' track will result in silence.

The '*Now Playing list*' is exposed as a folder item which may in turn be browsed with [ICinemoPlaylist](#). The contents of the '*Now Playing*' playlist will be an exact copy of the list of tracks currently stored in the Apple device's playback engine.

For iAP-2, the '*Now Playing list*' will only be available on devices running iOS version 8 or above. The basic version will only display UIDs of the contained tracks but no metadata. Metadata will be available (like with iAP-1 devices) if either the iAP-2 device is mounted in Cinemo MM or an implementation of the [ICinemolAPMediaLibraryAccess](#) interface was provided to the Apple device when calling [ICinemolAP::StartMediaLibraryInformation\(\)](#). Cinemo MM uses such an interface and will automatically connect it.

The '*Device Native Browse Hierarchy*' container may be empty when using iAP-2 protocol, since iAP-2 does not support native browsing of Apple devices. Like for the '*Now Playing*' playlist, an [ICinemolAPMediaLibraryAccess](#)

interface is required to enable this feature – the restriction for iOS 8 devices or higher does not apply here. If this interface is provided, Cinemo will try to reproduce the native browsing experience known from Apple devices. This happens automatically when the iAP-2 device is mounted in Cinemo MM.

The device hierarchy is exposed as a folder item which may in turn be browsed with [ICinemoPlaylist](#).

It is possible to display additional entries in the browse hierarchy, which contain all tracks below the parent entry. This feature can be enabled by adding the URL parameter “expand” to the browse URL. The argument of this parameter will be used as the name of the folder, which can be identified by the `CINEMO_VFS_FLAG_EXPANSION` flag of the `CINEMO_VFS_TYPE`.

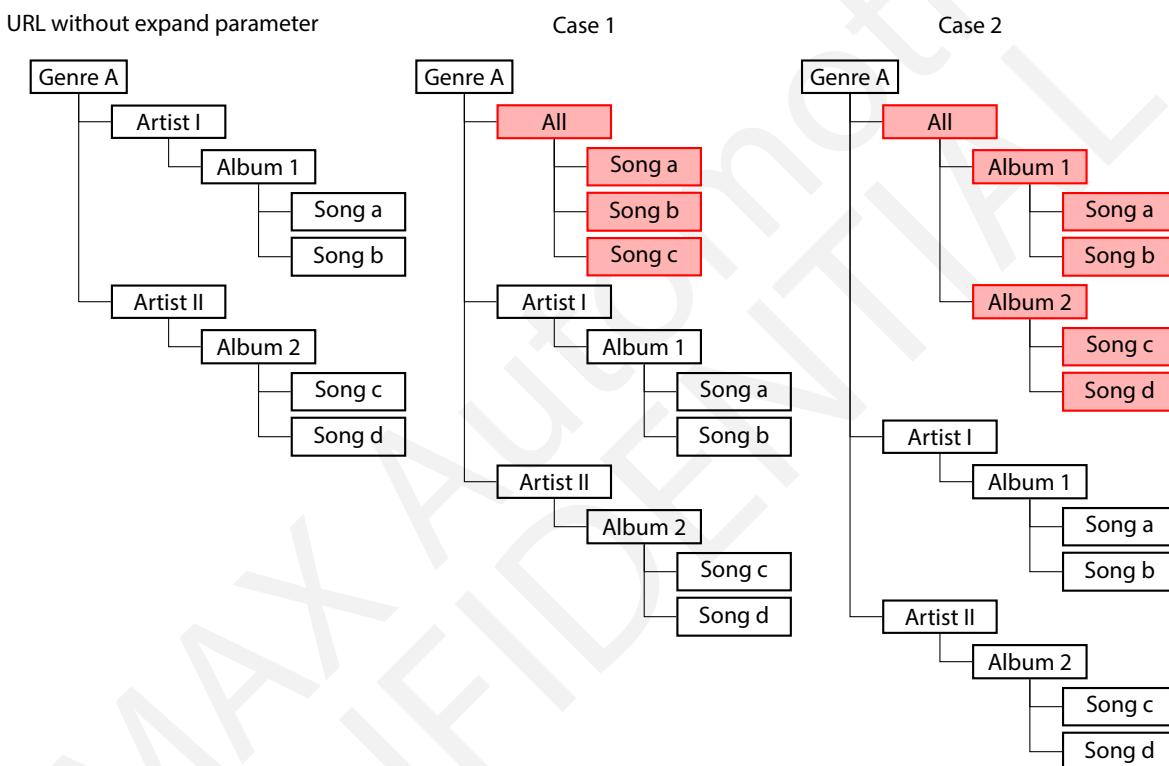


Figure 1.1: Overview of the different expand hierarchy options for iAP device browsing

There are two possibilities to display the expanded folder. Either as a flat hierarchy or as a folder with the regular Apple device hierarchy. Figure 1.1 shows the difference between both options. The flat hierarchy, Case 1 in Figure 1.1, can be enabled with the “expand=All&expand_type=0” URL parameter. The regular hierarchy, Case 2 in Figure 1.1, is enabled with “expand=All&expand_type=1”.

If the “expand” URL parameter is set for a certain URL, Cinemo will add the same parameter to all child containers, which are also expandable. For “expand_type=0” containers the expansion containers for a lower hierarchy level may be disabled by manually removing the “expand” parameter from the URL. For “expand_type=1” the the expansion containers for a lower hierarchy level cannot be disabled manually.

1.7.4 Playback

Although in principle any track in [ICinemoPlaylist](#) can be played by means of [ICinemoPlayer2::PlayTrack\(\)](#), it is recommended for Apple devices to play only the ‘Now Playing’ track – that is, the first track in the [ICinemoPlaylist](#) obtained by opening the root hierarchy of the Apple device. The ‘Now Playing’ track is a live stream which renders in real-time the audio and metadata streamed from the Apple device.

The reason for restricting playback to the ‘Now Playing’ track is because this more accurately reflects the device playback model. Apple devices don’t really play *tracks* at all: instead, they continuously output a *live stream* of audio and metadata, the control of which is achieved by iAP protocol commands which asynchronously select tracks for playback. Since the metadata for the currently playing track, along with the control commands which selected it, are independent from and asynchronous to the live audio stream produced by the device, it is impossible to determine the exact point in the audio stream where one track is stopped and another is started.

The calling application is supposed to maintain a single [ICinemoPlayer2](#) interface which continuously renders the live stream from the Apple device, and which will inform of player updates by means of the Cinemo event interface. In parallel, the calling application may support browsing of the Apple device native hierarchy using standard [ICinemoPlaylist](#) API, but instead of playing the resulting tracks, the application should call [ICinemoPlaylist::Select\(\)](#) or [ICinemoPlaylist::SelectTrack\(\)](#) to select them for playback on the Apple device. In this case, assuming the selection is successful, the application’s [ICinemoPlayer2](#) instance playing the ‘Now Playing’ track will seamlessly switch to playing the selected tracks.

Apple devices may enter a playback state, where there is actually no audio streamed to a connected accessory. This can be the case if the end of a playlist is reached and repeat is turned off. Cinemo will report such a state with a *CINEMO_EC_PLAYLIST_EMPTY* event. If such an event is received with the valid flag set to true, it is up to the connected accessory to refill the “NowPlaying” playlist of the iAP device. While iOS devices may accept a simple “Play” command to resume playback from the “All Tracks” list, older Apple devices like e.g. iPod nano 1G do require a [ICinemoPlaylist::Select\(\)](#) or [ICinemoPlaylist::SelectTrack\(\)](#) command to repopulate the “NowPlaying” playlist.

1.7.5 Playback Control from the Apple Device

On some Apple devices, the user is not prevented from interacting with the device while it is connected, and playback may therefore be controlled by the user interacting with the device itself. There are two kinds of playback control events which may occur:

- Playback control of the current track
- Device initiated playback of a completely different track

All playback control operations which originate from the Apple device – for example, PAUSE, RESUME, FFWD, FBWD – will be notified to the calling application by means of Cinemo playback events exactly as if the playback control had been initiated by its corresponding [ICinemoPlayer2](#) method. The calling application must therefore be prepared to receive events notifying a change of state not requested by the application itself.

A user interacting with the Apple device may initiate playback of a completely different track to the one currently playing. In this case, The [ICinemoPlayer2](#) instance playing the ‘Now Playing’ track will simply continue

to stream the new track selected on the Apple device, and *CINEMO_EC_METADATA* events will be posted to indicate a change to the ‘now playing’ metadata. The current track time and duration will also be updated.

Please note that Seek() is supported for all iAP-1 devices while iAP-2 devices support Seek() starting with iOS 8.

1.7.6 iAP Interface

Cinemo exposes an iAP interface [ICinemolAP](#) which allows an application to communicate with an Apple device opened by Cinemo. This interface should be used if the application wishes to access a feature of the device which is not directly related to Cinemo Media Engine (i.e. a non-multimedia feature).

In order to use [ICinemolAP](#) interface, the following steps must be followed:

1. Create an instance of [ICinemolAP](#) for the Apple device by calling [CinemoCreateIAP\(\)](#), specifying the URL of the device obtained from USB Discovery or other means. This must be done before opening the device with any other Cinemo API, e.g. [ICinemoVFS](#) or [ICinemoPlaylist](#), because the application must identify itself as an accessory with application-specific feature requirements.
2. Open the iAP device by calling [ICinemolAP::Open\(\)](#). Optionally pass custom identification parameters with the [ICinemolAP::Open\(\)](#) call. The returned attributes will contain information whether the device is using iAP-1 or iAP-2.
3. If the device is subsequently opened for browsing or playback with [ICinemoVFS](#) or [ICinemoPlaylist](#) then Cinemo Media Engine will also communicate with the device in parallel, using only the multimedia features of the device.

If no customized identification was used with an instance of the [ICinemolAP](#) interface, then a default identification tailored to the needs of the Cinemo Media Engine is performed as soon one of the interfaces [ICinemoPlaylist](#) or [ICinemoVFS](#) is created and opened.

1.7.7 Indexing iAP devices

Apple devices may be indexed using the Cinemo Media Management Engine either with the iAP-1 or iAP-2 protocol. However, due to performance and usability issues some iAP-1 devices may not be indexed and only be used for regular playback.

It is possible to determine if an Apple device may be indexed using the iAP-1 protocol by looking at the metadata of such a device:

```
# ./SampleMetadata -d "iap://usb:///dev/bus/usb/003/012"

opening...
path: iap://usb:///dev/bus/usb/003/012
type: FOLDER | VIRTUAL | ORDERED | WATCHABLE | SELECTABLE | USB | IAP | iAP-1
title 00 index 00 VFS_NAME: "Cinemo's iPod nano 5G"
title 00 index 00 DeviceIdentificationInfo: "5U9433XZ726"
title 00 index 00 VFS_UUID: "5U9433XZ726_iAP1"
title 00 index 00 VFS_IAP1_IPOD_VERSION: "1.0.2"
title 00 index 00 VFS_IAP1_LINGO_OPTIONS: 24575
title 00 index 03 VFS_IAP1_LINGO_OPTIONS: 3
```

```
title 00 index 04 VFS_IAP1_LINGO_OPTIONS: 23
title 00 index 10 VFS_IAP1_LINGO_OPTIONS: 1
title 00 index 00 VFS_MAY_BE_INDEXED: 0
title 00 index 00 VFS_IAP_FLAGS: (4000000h) BROWSING_HIERARCHY_AVAILABLE
...
```

The VFS_MAY_BE_INDEXED metaname will be set to either 1 or 0, where 1 means that Cinemo MM can index this device.

For iAP1 devices there are three different categories regarding indexing:

- **Really old devices**

These devices don't support the iAP protocol yet, they only offer a Mass Storage Device interface. Cinemo Media Management can index these devices like any other regular Mass Storage Device without any restrictions.

Examples: iPod classic 4G, iPod mini 1G and 2G

- **Old devices**

Most first or second generations of Apple devices support an early version of the iAP1 protocol, which lacks a key feature required for indexing. This so-called "database engine" is described in detail in the MFi Specification. Cinemo Media Management does not support indexing of these devices.

Examples: iPod nano 2G, iPod touch 2G, iPhone 3G

- **More recent devices**

In later versions of the iAP1 protocol the database engine has been added, which is used by Cinemo for indexing. It allows background operations without interfering with the currently playing track. Using this feature, Cinemo can index these devices efficiently.

Examples: iPod nano 7G, iPod touch 4G, iPhone 4

1.7.8 Native HID support

Certain iAP2 transports, like bluetooth or USB host mode, do require sending native HID reports instead of HID reports via iAP2 protocol. Cinemo can send native HID reports if the underlying transport library implementation does support this feature.

All native HID components need to be registered with the iAP device using the regular identification metapool of [\[CinemoIAP::Open\(\)\]](#). For Cinemo to be able to use native HID reports, the HID report descriptor of at least one "media playback remote" needs to be provided in this metapool.

1.7.9 EAP support

Cinemo has full support of the external accessory protocol (EAP) feature of the Apple devices. Both the native EA protocol as well as the iAP EA protocol features are supported. The native EA protocol support is described in the section [Native EAP support](#).

This section focuses on the iAP EAP support (also called "normal EAP"). Only this kind of EAP support can handle multiple EAP sessions for the same protocol. Therefore Cinemo requires that (for iAP2) the send message "StatusExternalAccessoryProtocolSession", which according to Apple Accessory Interface specification

is optional, to be identified with the Apple device. Cinemo Media Engine does not enforce this, but will produce an iAP2 protocol violation (which will be detected by the certification tools from Apple Inc.). The sample code for iAP identification configuration always adds this message to the identification settings, because this message is also required to allow the user of the Cinemo API to request the closing of an EAP session.

Normal EAP sessions will be announced via the [ICinemoIAPEAPAnnounce](#) interface, which can be provided to an instance of [ICinemoIAP](#) at Apple device connection time.

1.7.10 Native EAP support

Apple devices capable of iAP2 which are connected in USB host mode support native EAP sessions. Usually these sessions have higher data rates when compared to normal EAP sessions since they don't have the iAP protocol overhead. Cinemo does support native EAP sessions if the underlying transport library implementation does support this feature.

Native EAP sessions don't show up in the same way as the normal EAP sessions do. They are *not* announced via the [ICinemoIAPEAPAnnounce](#) interface.

All native EAP sessions need to be configured in the identification metapool of [ICinemoIAP::Open\(\)](#) with the `CINEMO_METANAME_IAP_EAP_NATIVE_TRANSPORT_ID` set accordingly. If the setup is correct, each native EAP session will show up as an individual entry in the device metapool.

```
===== Device Metapool =====
...
[ 1|00] VFS_PATH: "iaptrack:///usb:///dev/bus/usb/002/003"
[ 1|00] VFS_NAME: "NowPlaying"
[ 1|00] VFS_TYPE: 883164449 0x34A40521
[ 1|00] VFS_NOWPLAYING_TYPE: 2 0x00000002
[ 2|00] VFS_PATH: "iap:///usb:///dev/bus/usb/002/003?pid=0"
[ 2|00] VFS_NAME: "NowPlaying"
[ 2|00] VFS_TYPE: 883164770 0x34A40662
[ 2|00] VFS_NOWPLAYING_TYPE: 1 0x00000001
[ 3|00] VFS_PATH: "iap:///usb:///dev/bus/usb/002/003?pid=1"
[ 3|00] VFS_NAME: "Cinemo's iPhone 6 HW-794"
[ 3|00] VFS_TYPE: 883164770 0x34A40662
[ 4|00] VFS_PATH: "iapneap://usb:///dev/bus/usb/002/003?pid=1"
[ 4|00] VFS_NAME: "com.cinemo.test1"
[ 4|00] VFS_IAP2_NATIVE_EAP_SESSION_STATE: 0 0x00000000
[ 4|00] VFS_TYPE: 883163234 0x34A40062
...
=====
```

In this example the native EAP session is listed as entry 4. It has a `VFS_NAME` which corresponds to the registered EA protocol name. The `VFS_IAP2_NATIVE_EAP_SESSION_STATE` will reflect the state of this session with the `CINEMO_IAP_NATIVE_EAP_SESSION_STATE` enum defined in `cinemo_iap.h`.

To communicate with an iOS application via this native EAP session, a [ICinemoDataPort](#) object has to be created using the provided `VFS_PATH`. This, however, will only be possible if the session state is *online*. Only iOS

applications may start an EAP session, so it may be necessary to call [ICinemoIAP::RequestAppLaunch\(\)](#) to bring a session *online*.

HUMAX Confidential

1.8 CarPlay

Cinemo Media Engine's CarPlay solution constitutes a full implementation of Apple's [CarPlay](#) client protocol. This consists of all IP-layer functionality including Apple's communication plugin, audio and video streaming. While this chapter provides a general overview of Cinemo's CarPlay solution, the interface itself is documented in more detail in the section [ICinemoCarPlay](#).

1.8.1 Architecture

Cinemo's implementation of CarPlay is modular and exposed by multiple Cinemo interfaces, [ICinemoCarPlay](#) and [ICinemoPlayer](#), as depicted by blue boxes in the diagram below. This separation allows the CarPlay client to run independently of [ICinemoPlayer](#) instances which may render the resulting video stream.

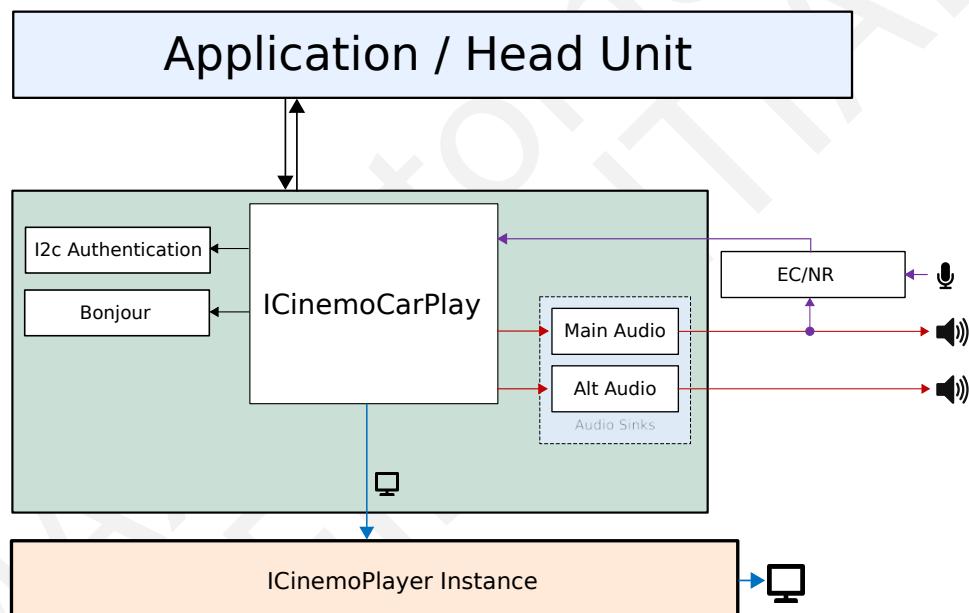


Figure 1.2: Carplay Architecture

1.8.2 CarPlay Client

[ICinemoCarPlay](#) exposes CarPlay client core functionality. It contains an implementation of Apple's CarPlay Communication Plugin. It supports low-latency audio rendering and capture. It does NOT render video itself, but it can stream compressed video by means of a Cinemo proprietary protocol to a [ICinemoPlayer](#) instance.

[ICinemoCarPlay](#) exposes a local control and callback interface. The control interface supports all known CarPlay protocol commands which may be transmitted from the accessory to the Apple device. The callback interface implements all known events and state notifications which may be transmitted from the Apple device to the

accessory. It is not required for an application to implement all callback functions; it may register only for the events it wishes to receive.

1.8.3 Audio Playback and Input

In order to achieve minimum latency, [ICinemoCarPlay](#) supports *only* local rendering of audio. It can render main and alternate audio streams directly using Cinemo's built in low-latency audio output device implementations specific to the target platform (e.g. ALSA, io-audio). Configuration of audio device parameters is exposed by [ICinemoCarPlay](#) interface methods similar to those of [ICinemoPlayer](#).

For audio input, [ICinemoCarPlay](#) can be configured to use a built-in low-latency audio capture implementation specific to the target platform (e.g. ALSA, io-audio). The current implementation assumes that Echo Cancellation / Noise reduction is performed as part of the capturing process outside of the Cinemo domain, so that an already cleaned audio signal is provided to Cinemo via the used capturing API.

1.8.4 User Events

User input devices are represented as Human Interface Devices (HID) and user events are transmitted from the accessory to the Apple device in the form of HID reports.

Applications may send HID reports by [ICinemoCarPlay::SendHIDReport\(\)](#), for example, to implement multi-touch pads, knobs and buttons. Cinemo passes HID descriptors and reports directly to the Apple device as raw binary data.

1.8.5 Location Information

Location information in the form of National Marine Electronics Association (NMEA) sentences can be provided to an Apple device using the [ICinemolAP::SendLocationInformation\(\)](#) method.

To enable support for sending location information, the support for Location Lingo (iAP1) or the LocationInformation command (iAP2) needs to be enabled during identification to the Apple device. An example of how to perform custom identification to an Apple device using the Cinemo API can be found in the SamplelAP source code.

In order to send location data with the Cinemo API, an implementation of the [ICinemolAPLocationInfo](#) interface needs to be registered when calling [ICinemolAP::Open\(\)](#). As soon as the Apple device requests location data, this interface will be used to signal the start of a session. From this time on, location data needs to be sent regularly to the Apple device using [ICinemolAP::SendLocationInformation\(\)](#). If the stop method of the interface is called, no further location data must be sent to the Apple device.

1.8.6 Vehicle Status

Vehicle Status information can be provided to an Apple device using the iAP2 protocol via the [ICinemolAP::SendVehicleStatus\(\)](#) method. For iAP1 devices vehicle status information is currently not supported by Apple.

To enable support for sending vehicle status information the `VehicleInformationComponent` and `VehicleStatusComponent` (iAP2) need to be enabled during identification to the Apple device. An example of how to perform custom identification to an Apple device using the Cinemo API can be found in the `SampleIAP` source code.

In order to send vehicle data with the Cinemo API, an implementation of the `ICinemolAPVehicleStatus` interface needs to be registered when calling [ICinemolAP::Open\(\)](#). As soon as the Apple device requests vehicle status data, this interface will be used to signal the start of a session. From this time on, vehicle data needs to be sent to the Apple device using [ICinemolAP::SendVehicleStatus\(\)](#) at intervals defined in the MFi spec. If the stop method of the interface is called, no further vehicle status data must be sent to the Apple device.

1.8.7 CarPlay and iAP-2 Protocol

Apple's CarPlay protocol is purely a *network* protocol, and therefore may run independently of any USB or iAP specific considerations. For example, CarPlay over WiFi network is possible without a USB connection. The only requirement for running [ICinemoCarPlay](#) is that network connectivity is established with the Apple device, so that it can, for example, receive and respond to multicast Bonjour advertisements.

However, Apple CarPlay protocol is expected to be run in parallel with an established iAP-2 session, which is used to obtain information, for example, about the 'Now Playing' track in order to drive a secondary display.

Due to the separate nature of iAP and CarPlay protocols, the calling application is free to choose from various implementations. For example, an application may wish to use Cinemo's iAP implementation, but implement the CarPlay protocol itself using Apple's communication plugin. On the other hand, it may wish to use Cinemo's CarPlay implementation, but implement the iAP protocol itself.

1.8.8 iAP and USB Host Mode

This section assumes that iAP-2 connectivity with the Apple device will be established using Cinemo Media Engine. If the Apple device is connected by USB, then it must be connected to USB hardware capable of role reversal, and an iAP-2 session must be established as described in [USB Host Mode](#). The following steps should be taken before instantiating [ICinemoCarPlay](#):

1. The Apple device is connected.
2. The Apple device, in USB device mode, is detected as described in [USB Discovery](#).
3. The Apple device is role-reversed as described in [USB Host Mode](#).
4. The Apple device, now in USB host mode, is assigned an IPv4 or IPv6 address.

Note the last step must be performed by the calling application, not by Cinemo. This step depends on the Apple device successfully enumerating the USB NCM interface exposed during role-reversal, and the correct mapping of the NCM interface to an NIC by the operating system kernel. Cinemo, which runs in user-space, cannot be responsible for these steps.

After these steps have been performed, it will be possible to run [ICinemoCarPlay](#) to instantiate a CarPlay session, and simultaneously, [ICinemoVFS](#), [ICinemoPlaylist](#) or [ICinemolAP](#) to handle the iAP-2 session. For example,

[ICinemoVFS](#) may be used to obtain metadata about the ‘Now Playing’ track. For other kind of Apple device features the additional functionality provided through the [ICinemoIAP](#) interface shall be used.

1.8.9 Detecting CarPlay support

With recent iOS versions it is possible to detect if a device supports CarPlay. [Cinemo USB detection](#) does provide this information in the metapool.

```
path: usbdetect://usb:///dev/bus/usb/003/009
type: FOLDER | DEVICE | VIRTUAL | ORDERED | METADATA_COMPLETE | USB | IAP
title 01 index 00 VFS_PATH: "iap://usb:///dev/bus/usb/003/009"
title 01 index 00 VFS_NAME: "iPhone"
title 01 index 00 VFS_TYPE: FOLDER | DEVICE | VIRTUAL | ORDERED | METADATA_COMPLETE |
WATCHABLE | USB | IAP
title 01 index 00 VFS_USB_ID: 512
title 01 index 00 VFS_USB_DEVICE_CLASS: 0
title 01 index 00 VFS_USB_DEVICE_SUBCLASS: 0
title 01 index 00 VFS_USB_DEVICE_PROTOCOL: 0
title 01 index 00 VFS_USB_MAX_PACKET_SIZE: 64
title 01 index 00 VFS_USB_VENDOR_ID: 1452
title 01 index 00 VFS_USB_PRODUCT_ID: 4776
title 01 index 00 VFS_USB_DEVICE_ID: 1794
title 01 index 00 VFS_USB_MANUFACTURER: "Apple Inc."
title 01 index 00 VFS_USB_PRODUCT: "iPhone"
title 01 index 00 VFS_USB_SERIAL_NO: "dcc58bc1cd6f5aa561b17b7a12689de5aa0c7f7e"
title 01 index 00 VFS_USB_NUM_CONFIGURATIONS: 4
title 01 index 00 VFS_USB_CARPLAY_SUPPORTED: 1
title 00 index 00 VFS_INDEX: 1
title 00 index 00 VFS_COUNT: 1
title 00 index 00 VFS_TOTAL: 1
```

The metaname *CINEMO_METANAME_VFS_USB_CARPLAY_SUPPORTED* will indicate that the device did respond to the CarPlay command. If it is missing, the device does not support this command. The value of this entry represents the result of the command. A value of 1 means that CarPlay is supported, a value of 0 means that CarPlay is not supported or disabled on the device.

1.9 Android Auto

The Android Auto solution provided by Cinemo constitutes a full implementation of Google's [Android Auto](#) Projection (AAP) Receiver Library. This includes basic communication functionality, audio and video streaming as well as all extensions like navigation, media playback or sensor data. While this chapter provides a general overview of Cinemo's Android Auto solution, the interface itself is documented in more detail in the section [ICinemoAndroidAuto](#).

1.9.1 Architecture

Cinemo's implementation of the Android Auto Projection protocol is modular and exposed via multiple Cinemo interfaces. A general overview is shown in the diagram below.

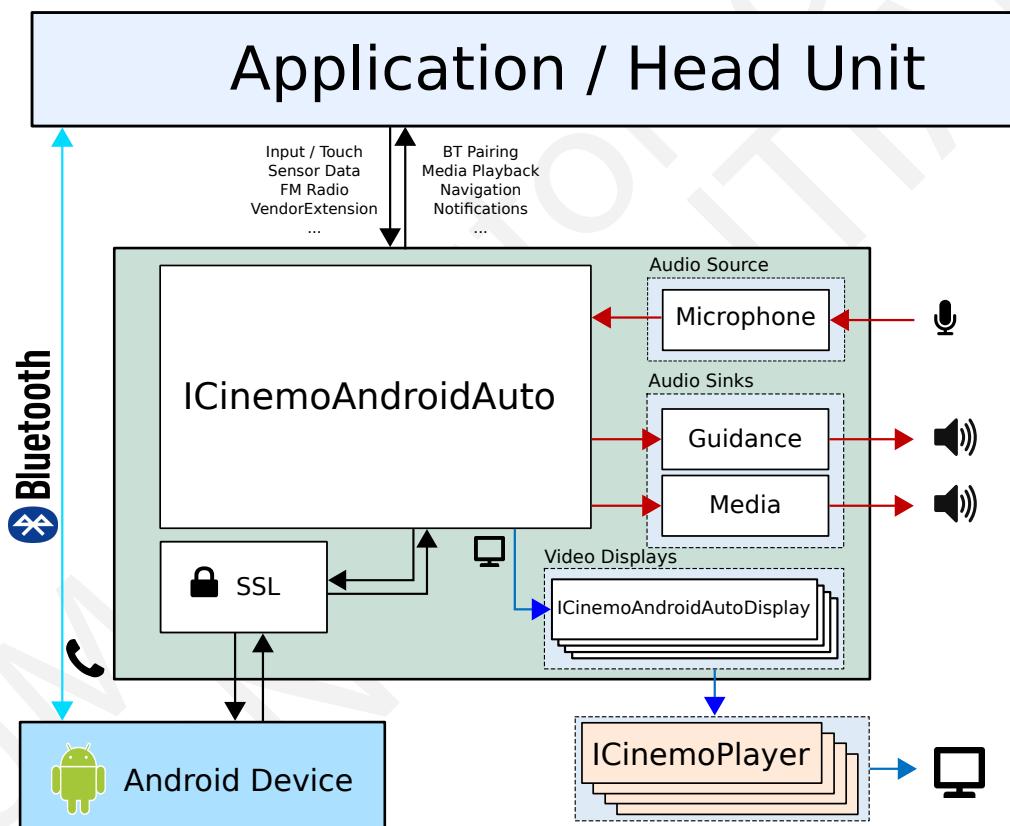


Figure 1.3: Android Auto Projection Architecture

The base interface, [ICinemoAndroidAuto](#), is used for controlling the session itself as well as the different audio streams. All remaining features are available on additional interfaces, e.g. audio capturing, all kinds of notifications or data exchange.

The display interface, [ICinemoAndroidAutoDisplay](#), provides all methods necessary for playing the available video streams and input event processing.

1.9.2 Android Auto Client

The [ICinemoAndroidAuto](#) interface exposes the Android Auto Projection core functionality. It allows access to Cinemo's integration of the Android Auto Projection Receiver Library and supports low-latency audio rendering and capture. Video playback is handled by playing dedicated [ICinemoAndroidAutoDisplay](#) objects via the [ICinemoPlayer2](#)/[ICinemoPlaylist](#) infrastructure.

[ICinemoAndroidAuto](#) exposes basic control mechanisms for all audio streams being transmitted from the Android device to the head unit and vice versa. The respective methods for the video streams are available on each [ICinemoAndroidAutoDisplay](#) object. Callback functions are used to signal events and state notifications received from the Android device. While some callbacks are essential for mandatory functionality (e.g. audio/video focus requests), it is not required for an application to implement all callback functions. It may register only for the events it wishes to receive.

Advanced features of the Android Auto receiver library are available on respective dedicated interfaces, which can be queried from the base [ICinemoAndroidAuto](#) object.

The device URL of the Android device which is currently used in an [ICinemoPlayer2](#) to run Android Auto is available in the metapool from the [ICinemoAndroidAuto::GetMetapool\(\)](#) function under the metaname CINEMO_METANAME_VFS_BIND_PATH. It is also passed in the szurl parameter of the CinemoAndroidAuto_NewConnection() callback.

1.9.3 Audio Playback and Input

In order to achieve minimum latency, Cinemo's Android Auto implementation only supports local rendering of audio. It can render all audio streams directly using Cinemo's built in low-latency audio output device implementations specific to the target platform (e.g. ALSA, io-audio).

For audio capturing Cinemo's built-in low-latency implementation specific to the target platform (e.g. ALSA, io-audio) will be used.

1.9.4 Input Events

Input events may be transmitted to the Android device in one of two ways:

- Applications may send input events directly to the Android device using the [ICinemoAndroidAutoDisplay](#) interface, which supports multi-touch, touch pads, knobs and buttons. Cinemo passes these events directly to the Android device as raw data.
- Connected [ICinemoPlayer2](#) instances register a "touchscreen" for transmitting single touches and a limited number of other user events which are exposed by [ICinemoPlayer2](#) interface methods used, e.g. to control DVD or Blu-ray playback applications.

This two-path approach gives an application great flexibility. It can achieve full and complete control of user input to the Android Auto session with the [ICinemoAndroidAutoDisplay](#) interface, or it can rely on Cinemo's default behavior, where connected [ICinemoPlayer2](#) instances will be fully operational and functionally equivalent to playing any other kind of interactive media source.

The advantage of the latter approach is simplicity and code re-use: applications using [ICinemoPlayer2](#) need not know if they are controlling a Android Auto session or a DVD menu.

1.9.5 Video display handling

For each video stream a distinct [ICinemoAndroidAutoDisplay](#) objects is required. These objects are initialized by calling the [ICinemoAndroidAuto::GetDisplay\(\)](#) method with an `index` parameter corresponding to the title value which was used to configure the display in the setup metapool.

It is possible to register as many video streams as Android Auto supports, which is four at the moment. There are two primary streams for the main and cluster display and two auxiliary streams which provide navigation and turn card data. The configuration of the video streams is also done via the metapool passed to the [ICinemoAndroidAuto::Setup\(\)](#) method. Both primary and auxiliary streams need a `CINEMO_METANAME_ANDROIDAUTO_VIDEO_DISPLAY_TYPE` entry, while auxiliary displays need a `CINEMO_METANAME_ANDROIDAUTO_VIDEO_AUXILIARY_DISPLAY_TYPE` entry as well.

When configuring the `CINEMO_METANAME_ANDROIDAUTO_VIDEO_DISPLAY_TYPE` metapool entry the possible values of `CINEMO_ANDROIDAUTO_DISPLAY_TYPE` are:

- `CINEMO_ANDROIDAUTO_DISPLAY_MAIN`
- `CINEMO_ANDROIDAUTO_DISPLAY_CLUSTER`
- `CINEMO_ANDROIDAUTO_DISPLAY_AUXILIARY`

The supported values for the `CINEMO_METANAME_ANDROIDAUTO_VIDEO_AUXILIARY_DISPLAY_TYPE` metapool entry are the `CINEMO_ANDROIDAUTO_AUXILIARY_DISPLAY_TYPE` enum.

Each primary video stream must only be registered once, while auxiliary displays may be registered multiple times.

Once the [ICinemoAndroidAuto::Setup\(\)](#) call returns, it is possible to access the display objects using the [ICinemoAndroidAuto::GetDisplay\(\)](#) method where the `index` parameter corresponds to the metapool title value for that display. The playback of a display object may be started at any time, even before the Android Auto session has been launched. As soon as a session starts the `CinemoAndroidAuto_VideoReady()` callback will be called for each video stream individually. After receiving this callback it is possible to set the video focus using the [ICinemoAndroidAutoDisplay::SetVideoFocus\(\)](#) method.

It is not possible to stop and later resume playback of the video streams. The playback may only be stopped if the associated video stream is not needed anymore for the remainder of the session.

1.9.6 Input handling

Each registered primary display handles input events separately, auxiliary displays are non-interactive and do not support input events. This requires to configure the available input devices for each display separately in the setup metapool. The metapool `title` values for the input configurations must match those of the dedicated display configuration.

1.9.7 ICinemoAndroidAuto life cycle

The Android Auto receiver library is agnostic to the underlying transport protocol and currently supports both wired and wireless connections with an Android device via USB or Wifi. Depending on the transport type the configuration, handling and life cycle of the [ICinemoAndroidAuto](#) and [ICinemoAndroidAutoDisplay](#) objects differ slightly. The following sequence diagrams show these differences in detail.

The term "session" in the following paragraphs is used to refer to a single connection between the mobile device and the head unit, programmatically enclosed by the `CinemoAndroidAuto_BeginSession()` and `CinemoAndroidAuto_EndSession()` callbacks.

Connection using USB

For wired connections the lifetime of the [ICinemoAndroidAuto](#) object matches the USB connection lifetime. As soon as a new Android device is discovered on the USB bus and Android Auto should be launched a new [ICinemoAndroidAuto](#) object needs to be created and opened using the corresponding USB device URL. When the [ICinemoAndroidAuto::Open\(\)](#) method is called the Android Auto session will be started, which results in the `CinemoAndroidAuto_BeginSession()` callback in case of success.

Both possibilities to end a session are shown in Figure above. The upper part shows the case where the session is ended by unplugging the device from the USB bus. In the lower part the call sequence for terminating a session from the head unit side is shown.

Connection using WiFi

For Android Auto wireless the objects involved in the process are the same as in the wired use case, however their behavior is different. When setting up a new [ICinemoAndroidAuto](#) object for the wireless use case it needs to receive an IP address and a port in the [ICinemoAndroidAuto::Open\(\)](#) call which will launch a TCP server, which listens for incoming connections from Android devices.

If a new connection is opened from a mobile device the `CinemoAndroidAuto_NewConnection()` callback will be called to signal this. Only if this callback returns [CinemoNoError](#) the connection will be accepted and an attempt to establish an Android Auto session will be started. If this attempt is successful the new wireless session will be announced using the `CinemoAndroidAuto_BeginSession()` callback. While a session is active the TCP socket will continue listening for new connections. The socket is kept open until the moment [ICinemoAndroidAuto](#) object is released or the [ICinemoAndroidAuto::Close\(\)](#) method is called. If an application wants to make sure that the socket is already available it needs to watch the [ICinemoMetapool](#) of the [ICinemoAndroidAuto](#) object. As soon as the socket has been opened the `CinemoAndroidAuto_SessionEvent()` callback will be called with the `CINEMO_ANDROIDAUTO_EVENT_METAPOOL_UPDATED` event parameter and the metapool will contain the `CINEMO_METANAME_ANDROIDAUTO_WIFI_SERVER_ADDRESS`. This metapool can be obtained from the [ICinemoAndroidAuto::GetMetapool\(\)](#) method.

In order to close a wireless session without shutting down the accepting TCP socket, the [ICinemoAndroidAuto::DisconnectSession\(\)](#) method can be used. When the [ICinemoAndroidAuto](#) object is closed using [ICinemoAndroidAuto::Close\(\)](#) method all active sessions will be closed automatically.

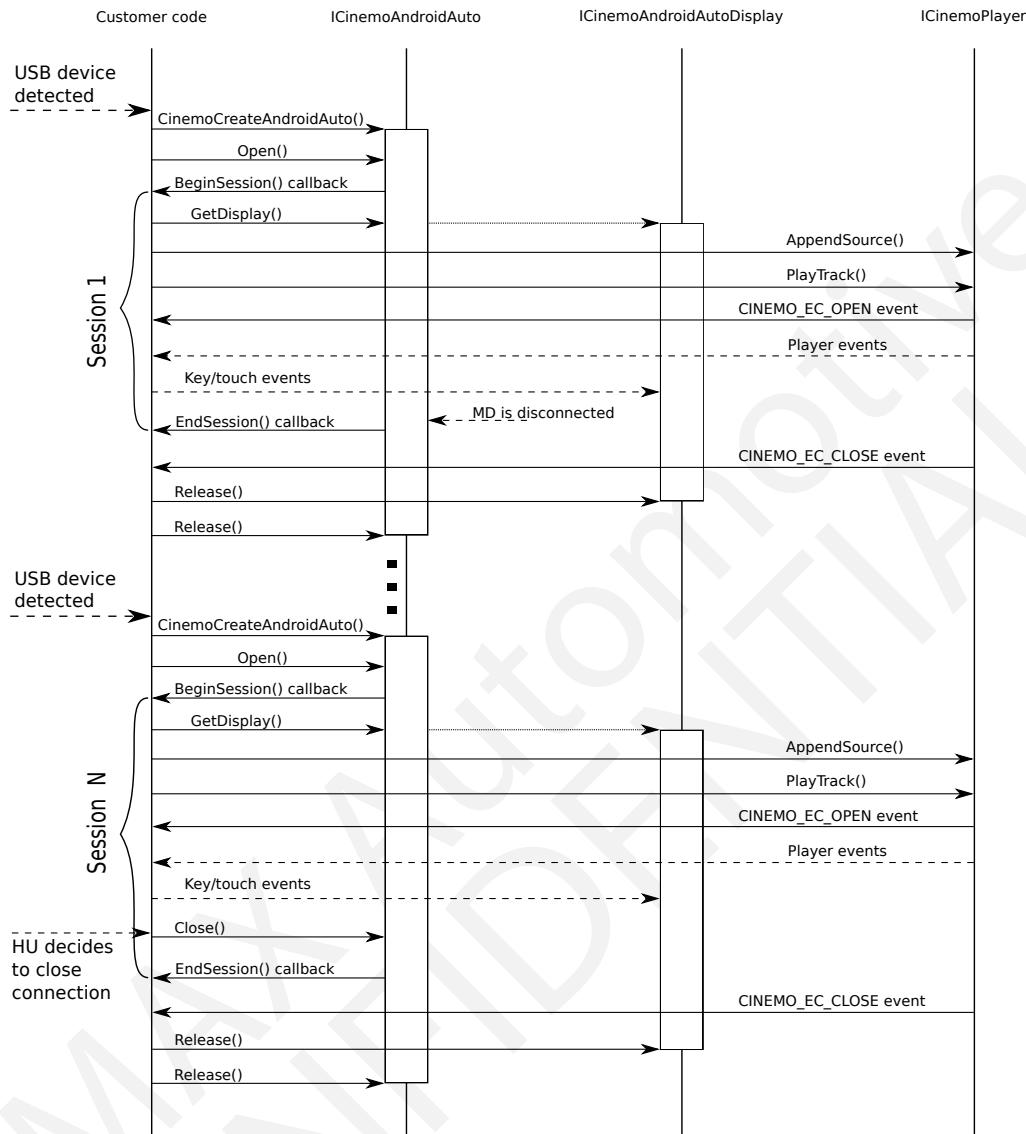


Figure 1.4: Call sequence for wired Android Auto connections

The major difference between the wired and wireless use case is the life cycle of the [ICinemoAndroidAuto](#) object. When connecting a mobile device using the USB bus, the life cycle can be synced with the USB connection. For wireless connections there needs to be an open socket to accept new connections, which corresponds to the lifetime of the [ICinemoAndroidAuto](#) object. Individual sessions can then be opened and closed independently from the [ICinemoAndroidAuto](#) object. However, it is only possible to have a single active connection at a time.

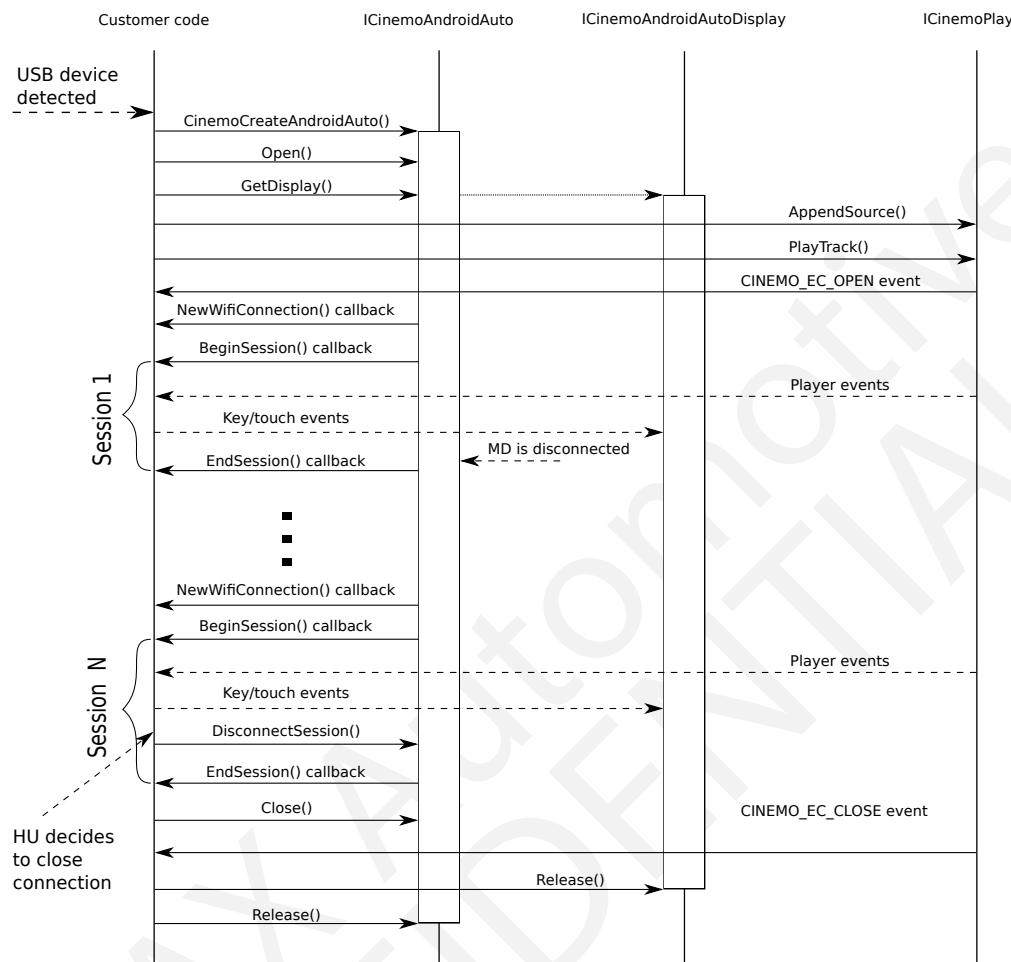


Figure 1.5: Call sequence for wireless Android Auto

1.10 Playback from Audio and Video capture devices

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Cinemo Media Engine supports playback from Audio and Video capture devices. This playback mode is designed for low delay and a zero copy capture to display pipeline and does not support Distributed Playback. Only live playback of the captured Audio and Video is supported. The playback cannot be paused and resumed.

Audio and Video sync is maintained by delaying audio samples and video frames by a configurable amount after their capture timestamp. The delay amount should be chosen as low as possible while ensuring that no sample or frame arrives late at the audio or video renderer. The delay amount can be configured by setting the **CINEMO_OPTION_BUFFER_CAPTURE_DELAY_MS** option. The minimal delay of the

audio playback is at least the sum of the two options `CINEMO_OPTION_AUDIO_DEVICE_BUFFER_MS` and `CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_PERIOD_MS`.

The capturing devices and options are set by `CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_ID` and `CINEMO_OPTION_VIDEO_CAPTURE_DEVICE_ID` options.

Capture is started by playing the special URL `capture://`.

1.10.1 Video Capture Parameters

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

General and capture device specific parameters are set by augmenting the capture device in `CINEMO_OPTION_VIDEO_CAPTURE_DEVICE_ID` with an URL query string. Valid parameters for video capture devices are listed in the following table.

Parameter	Description
<code>resolution=WidthxHeight (unsigned)</code>	capture resolution
<code>framerate=FPS (float)</code>	frames per second
<code>crop=TL_x,TL_y,BR_x,BR_y (unsigned)</code>	top left and bottom right corner of the active area
<code>drm_device=/dev/dri/renderD128 (string)</code>	path to DRM device node (Linux specific)

1.10.2 Audio Capture Parameters

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

General and capture device specific parameters are set by augmenting the capture device in `CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_ID` with an URL query string. Valid parameters for audio capture devices are listed in the following table.

Parameter	Description
<code>samplerate=rate (unsigned)</code>	sample rate
<code>channels=num_channels (unsigned)</code>	number of channels

1.10.3 Linux Capture Example

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Playback on a Linux system from a Video4Linux2 and an ALSA device

```
ICinemoConfig::SetOption(CINEMO_OPTION_BUFFER_CAPTURE_DELAY_MS, "120");
ICinemoConfig::SetOption(CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_ID,
    "hw:0,2?samplerate=48000&channels=2");
```

```
ICinemoConfig::SetOption(CINEMO_OPTION_VIDEO_CAPTURE_DEVICE_ID,  
    "/dev/video0&resolution=1920x1080&framerate=25&"  
    "drm_device=/dev/dri/renderD128");
```

HUMAX Confidential

2. Cinemo Media Management Engine™

Cinemo Media Management Engine extends the functionality of Cinemo Media Engine with content indexing, search and remote access functions.

Cinemo Media Management Engine, hereafter referred to as Cinemo MM, is designed from the ground up to provide the fastest possible access to all available media content on a device. Since Cinemo MM is built on top of standard Cinemo interfaces, it is capable of recognizing and indexing all media formats supported by Cinemo Media Engine. This encompasses a vast number of multimedia formats from MP3 to High Definition Video. All major container and codec types are supported.

Cinemo MM is capable of indexing media content from a wide range of mass storage or USB connected devices, and can also be used as a persistent searchable repository for application supplied media content and associated metadata. This means, for example, that applications may perform their own indexing and metadata extraction, and upload the resulting metadata to Cinemo MM.

Cinemo MM is fully customizable with an XML configuration file, as described in [Cinemo MM Configuration](#). Although optimized for use as a media server, it can be configured to store and hierarchically represent any metadata which may conceivably be supplied by an application. For example, Cinemo MM has been configured as a searchable repository for metadata ranging from *contact information* for an address book, to *EXIF attributes* for an image database.

Please contact your Cinemo representative if you have questions about additional use cases.

2.1 Content Directory Service

Cinemo MM is based on a client/server architecture and runs as a UPnP content directory service, or *media server*, supporting local or remote browsing, search and playback of indexed or application supplied media content by either Cinemo or third-party UPnP clients.

Although detailed knowledge of UPnP protocols is not required to integrate or use Cinemo MM and associated Cinemo interfaces, this chapter assumes some basic knowledge of UPnP architecture and terminology. The relevant specification is [UPnP Content Directory Service 1.01](#).

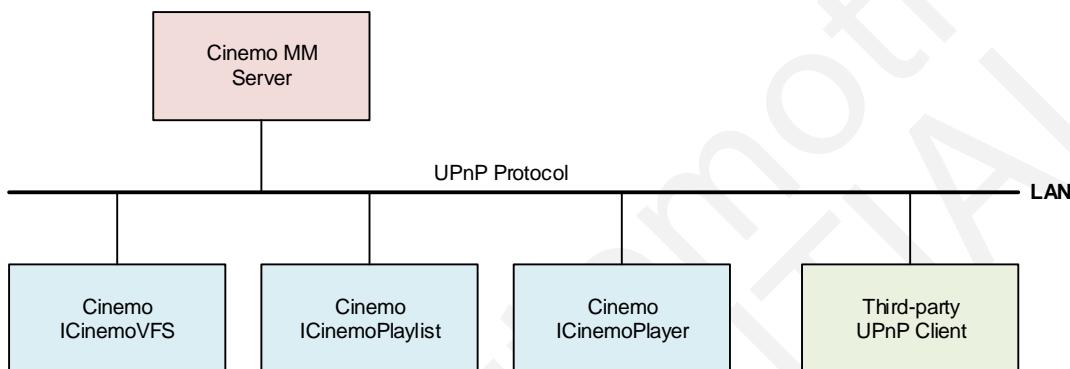


Figure 2.1: Cinemo MM server and clients

Cinemo MM is typically run as a single instance, and serves multiple clients. However, like all other Cinemo interfaces, there is no restriction on the number of Cinemo MM servers that can be instantiated. UPnP clients, on the other hand, can only connect to one server at a time. Clients cannot aggregate search or browse results from multiple servers.

2.1.1 UPnP Extensions

Some proprietary extensions to the UPnP protocol are implemented which provide access to optimized functionality by Cinemo clients – for example, when Cinemo clients browse Cinemo MM, a proprietary communication path eliminates the need to XML encode the browse results, while third-party clients will continue to receive XML in accordance with UPnP specifications. Cinemo clients may also access functions for mounting and dismounting volumes, journaling APIs, and metadata normalization by means of third-party media recognition services.

2.1.2 Cinemo Interfaces

All Cinemo interfaces related to browsing, searching and playback are ‘UPnP aware’. Both [ICinemoVFS](#) and [ICinemoPlaylist](#) can be used to discover generic UPnP media servers on the network, and browse or search the content on any media server. [ICinemoPlayer](#) can play content on any media server. [ICinemoMM](#), on the other hand, uses proprietary protocols and connects only to Cinemo MM. [ICinemoMM](#) serves a dual purpose: it can create a Cinemo MM server and administer it locally, or it can connect to a running Cinemo MM

server, and administer it remotely. Cinemo MM server can be administered by multiple [ICinemoMM](#) interfaces simultaneously.

All Cinemo interfaces communicate with UPnP media servers, Cinemo MM or otherwise, by means of a built-in optimized network stack. UPnP media servers are identified by a `upnp://` URL.

In accordance with Cinemo design goals, the network protocol between Cinemo interfaces and Cinemo or third-party UPnP media servers is transparent to applications using Cinemo API. For example, an application using [ICinemoVFS](#) or [ICinemoPlaylist](#) need not know if it is browsing a local or remote media server, or Cinemo MM, or even a local hard disk folder or USB device. There may be some behavioural differences which an application designer should be aware of, as described in following sections, but in principle an application using [ICinemoPlaylist](#) should be able to browse all kinds of object, UPnP or otherwise, using the same code.

2.1.3 Collation

Cinemo MM is delivered with a built-in heavily optimized implementation of the Unicode Collation Algorithm, providing intuitive sorting of browse and search results regardless of language, locale, and size of the result set. The Unicode Collation Algorithm will correctly handle the primary ignorable (e.g. non-alphabetical) characters often present in metadata extracted from media content, such that items are always ordered where they should be within the result set.

In accordance with [UPnP Content Directory Service 1.01](#) section 2.5.8, the sort criteria for each browse or search request is specified as part of the request. It is the responsibility of the UPnP media server to supply sorted results to the client. This is also a requirement of the fetch-on-demand strategy employed by Cinemo client interfaces: a result set cannot be sorted unless the result set is entirely known, but it would be prohibitively expensive for Cinemo clients to download the entire result set for sorting purposes. Therefore, sorting is performed server-side.

Cinemo MM is capable of sorting on all supported metadata fields or attributes. For optimal performance, Cinemo MM supports intelligent caching of search results, to handle cases where the same search request is submitted multiple times by one or more clients with different sort criteria: in this case, the results of a previous search are simply re-sorted according to the latest criteria.

2.1.4 Alphabetic Index

Cinemo MM search and browse results are provided with an alphabetic index, suitable for use by an alphabetical scrollbar in user interface designs. The alphabetic index supports all Unicode symbols and allows precise and fast navigation through large result sets.

The alphabetic index is generated on-the-fly by Cinemo MM for each result set and is delivered to [ICinemoVFS](#) and [ICinemoPlaylist](#) clients by means of a proprietary metadata extension. This is a requirement of the fetch-on-demand strategy employed by Cinemo client interfaces. A browse or search result set may consist of a very large number of tracks, which would be prohibitively expensive and wasteful of resources to download in entirety. However, the alphabetic index cannot be computed unless all results are known. Only Cinemo MM server is in a position to know the entire result set, therefore, the alphabetic index is generated server-side.

The alphabetical index may be obtained with [ICinemoPlaylist::GetAlphaIndex\(\)](#).

HUMAX Confidential

2.2 Items and Containers

Cinemo MM, like all UPnP media servers, exposes two classes of objects, *items* and *containers*. Items may be subclassed into audio, video or other kinds of playable or non-playable tracks. Containers may also be subclassed, for example, into albums, artists, filesystem folders, etc. The subclassing of items and containers is fully configurable as described in [Cinemo MM Configuration](#) – an application may configure additional classes for its own purposes, or it may disable the built in support for certain classes, e.g. audio or video tracks, by removing the corresponding class from the configuration.

Containers, unlike items, may be *browsed* or *searched*, but cannot be played. Browsing a container will return all immediate children of the container. Searching a container, on the other hand, will return all recursive descendants of the container which match a given query expression (it is possible to limit the depth of a search in the expression itself). In both cases, depending on the browse or search query, the result may consist of a mix of both items and containers.

2.2.1 Object IDs

All objects exposed by Cinemo MM are uniquely identified by a 64-bit integer ID. As long as database files are not deleted, **object identifiers will be fully persistent from one run to the next**. This applies not only to tracks or other content items, but also to container objects such as artists and albums – for example, if a container of type *artist* with title *Madonna* was once seen with a certain identifier, it is guaranteed to always have the same identifier in future.

This is an important constraint, since it enables an application using Cinemo MM to manage *playlist persistence* by simply remembering the ID of the container it was browsing, or the query expression it was searching. Since all browse and search query parameters, including the container ID, are encoded in the URL passed to [ICinemoPlaylist::Open\(\)](#) method, an application may simply store this URL, enabling it to re-create the playlist in future. Note, if an application wishes to use Cinemo MM database functionality to persistently store URLs or other types of arbitrary data, this is possible using persistent virtual volumes, described later in this chapter.

2.2.2 Fields

All item and container objects have associated [Fields](#). Some fields are common to all objects, for example, the object ID and class, while other fields are specific to the object itself, or specific to an application, since custom metadata fields can be defined in [Cinemo MM Configuration](#). There can be multiple values for each field, described in [Indexed Fields](#). All fields for an item or container object can be written to or modified by [ICinemoMM::EditNode\(\)](#). This applies to all objects within indexed or virtual volumes.

All fields for an item or container object are accessed as metadata with the [ICinemoMetapool](#) interface, which can be obtained in various ways. The following sections describe how to obtain metadata using [ICinemoPlaylist](#) and [ICinemoVFS](#) interfaces.

2.2.3 Accessing Objects and Fields

Objects and fields can be accessed by issuing queries to the Cinemo MM server. The general form of a query consists of the following URL:

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml
```

Additional URL parameters define which objects and fields shall be returned as query result.

Cinemo MM supports two kinds of URL parameters: URL parameters which map directly to UPnP arguments and thus can be interpreted by any UPnP server and URL parameters that are specific to the Cinemo MM server.

The following lists give an overview of supported URL parameters.

UPnP query arguments

This set of parameters of the URLs map to the UPnP arguments as described in the following list.

- **pid**

Defines the ID of the parent node. Maps to ObjectID argument for browse and ContainerID for search queries.

- **browse**

Maps to BrowseFlag on browse queries and is ignored on search.

- **search**

Defines the search query, maps to SearchCriteria argument.

- **sort**

Defines the sorting criteria, maps SortCriteria argument.

- **filter**

Defines the fields to be included in the result. Maps to Filter argument. Defaults to "*".

- **index**

The start index - first item returned. Maps to StartingIndex.

- **count**

Number of result returned. Maps to RequestedCount argument.

Further details on the functions and arguments can be found on:

<http://upnp.org/specs/av/UPnP-av-ContentDirectory-v1-Service.pdf>

Cinemo extensions

Cinemo MM extends several parameters in order to support a wider range of applications than UPnP. The extensions are summarized in the following list.

- **search**

The search features a powerful expression parser supporting an extended [Cinemo MM query language](#).

- **sort**

The SortCriteria argument can include expressions formulated in the [Cinemo MM query language](#) preceded by '\$'.

Cinemo specific parameters

This set of parameters of the URLs are Cinemo MM specific and will be ignored by third party UPnP servers.

- **thumb**

Defines the thumbnail selection strategy for the query. Overrides [CINEMO_OPTION_MM_THUMB_DEFAULT_STRATEGY](#).

- **expr**

Filter items in BrowseDirectChildren queries. Use an expression in [Cinemo MM query language](#) format.

- **candidates**

Enables next character candidates for text queries. See [ICinemoPlaylist::GetCandidate\(\)](#) API.

- **vars**

Enables all available result context vars to be returned as [CINEMO_METANAME_VFS_UPNP_TEXT_SEARCH_VARS](#).

See [Cinemo MM Sorting](#) for available vars.

The following sections provide several examples on the usage of URL parameters.

2.2.4 ICinemoPlaylist

Containers are typically browsed or searched with the [ICinemoPlaylist](#) interface, and browse or search results are returned as individual *tracks* in the resulting playlist. The 64-bit `track_id` of each [ICinemoPlaylist](#) track is equal to the corresponding Cinemo MM object ID. Note that not all tracks are playable, since some ‘tracks’ may actually be child containers, or other configured classes of object. The metadata returned in [ICinemoMetapool](#) for each track will indicate the object class, and whether or not the track is playable. Note that [ICinemoPlaylist](#) can always be played, regardless of its contents: [ICinemoPlayer](#) will simply skip over any non-playable tracks.

[ICinemoPlaylist](#) can be used to browse an object – that is, return a list of the object’s children – with the following URL. This will create a playlist with zero or more tracks, where each track corresponds to a child of the object. The count of tracks in the playlist will equal the number of children. This will only work for container objects.

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?
pid=<id>&
browse=BrowseDirectChildren
```

[ICinemoPlaylist](#) can be used to search an object – that is, return a list of the object’s descendants matching a specific query expression – with the following URL. This will create a playlist with zero or more tracks, where each track corresponds to an object which matches the query. The count of tracks in the playlist will equal the number of matches. This will only work for container objects.

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?
pid=<id>&
search=<query>
```

[ICinemoPlaylist](#) can also be used to obtain metadata about a single object, where the object’s ID is known, but its metadata is not. The following URL will open a playlist with a single track, where the track’s metadata corresponds to the object itself. This will work for both item and container objects.

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?
pid=<id>&
browse=BrowseMetadata
```

In all cases, **<addr:port>** is the IP address and port number of the media server, which may be known in advance or obtained by SSDP discovery, and **<id>** is the object ID. The ID of the Cinemo MM root node is 0 (zero), in accordance with UPnP specification.

2.2.5 ICinemoVFS

[ICinemoVFS](#) can also be used to access the metadata for an object, or for a container object's children, depending on the URL which is passed to [ICinemoVFS::Open\(\)](#) method. Note, it is acceptable, and sometimes simpler, to use [ICinemoVFS](#) for obtaining metadata about an object, but it is not recommended to use [ICinemoVFS](#) for obtaining the children of a container object unless there are good and known reasons for doing so. If this is not the case, then it is strongly recommended to use [ICinemoPlaylist](#) instead, which incorporates many enhancements, including fetch-on-demand strategies for containers with large numbers of children. [ICinemoPlaylist](#) also offers the advantage that the resulting tracks can be played.

The following URL will return an [ICinemoMetapool](#) with a single track, where the track's metadata corresponds to the object itself. This will work for both item and container objects.

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?
pid=<id>&
browse=BrowseMetadata
```

The following URL, on the other hand, will return an [ICinemoMetapool](#) containing zero or more tracks, where each track corresponds to the immediate child of the container object being opened. The count of tracks in the [ICinemoMetapool](#) will equal the number of children, which could be very large and consume significant system resources. This will only work for container objects.

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?
pid=<id>&
browse=BrowseDirectChildren
```

The following URL is a safer version of the above, specifying the start index and maximum count of tracks to return. This will only work for container objects. This is essentially what [ICinemoPlaylist](#) is doing in the background, in order to implement its fetch-on-demand strategy.

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?
pid=<id>&
browse=BrowseDirectChildren&
index=1&
count=100
```

In all cases, <addr:port> is the IP address and port number of the media server, which may be known in advance or obtained by SSDP discovery, and <id> is the object ID. The ID of the Cinemo MM root node is 0 (zero), in accordance with UPnP specification.

2.2.6 Change Notifications

Cinemo MM, unlike most third-party media server implementations, supports real-time notification of content change from server to client. For example, if an application uses [ICinemoPlaylist](#) to browse or search the 'All Music' container of an MM server, and if a mass storage device containing music tracks is subsequently mounted or dismounted on the MM server by some other application, process or thread, the changes will be immediately and automatically reflected in the playlist contents.

An application using [ICinemoPlaylist](#) or [ICinemoVFS](#) should therefore be aware that it may often be chasing a moving target, and that change notifications may occur relatively frequently, especially when new content is being indexed by Cinemo MM. This is an important design consideration which cannot be simply ignored. Events posted by the [ICinemoPlaylist](#) instance will notify the application when such changes have occurred, and are limited by UPnP specification to a maximum frequency of 0.5Hz. The [ICinemoVFS::Watch\(\)](#) method can also be used to obtain updates.

To provide the best possible browsing experience, an application should take care to update its user interface to reflect changes in playlist contents *if and when* they occur. This is the only responsibility of the application, since everything else will be handled internally by Cinemo – for example, [ICinemoPlayer](#) will handle the disappearance of its currently playing track from a playlist, should this occur, by automatically advancing to the next available track.

Note that this behaviour is not specific to browsing Cinemo MM server – in practise, any kind of object supported by [ICinemoPlaylist](#) or [ICinemoVFS](#) may generate change event notifications. This includes but is not limited to SSDP discovery, browsing iAP or MTP devices, and perhaps when supported in the future, local folders on mass storage devices. All browsable content is subject to change.

2.3 Volumes, Volume Groups and Hierarchies

Cinemo MM supports three container categories: *volumes*, *volume groups*, and *hierarchy nodes*. The entire UPnP browse hierarchy, consisting of possible all items and containers, is formed out of these containers and their children.

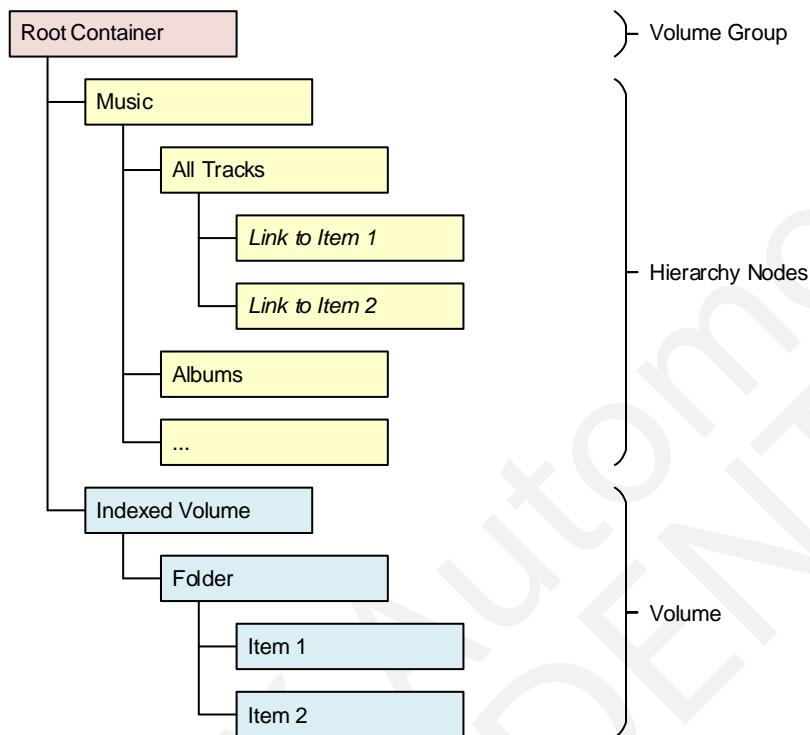


Figure 2.2: A simple UPnP browse hierarchy

Volumes can contain anything, but usually items and associated metadata, which are either indexed by Cinemo MM (an indexed volume), or supplied externally by an application (a virtual volume). All items must reside in a volume. There is nowhere else for an item to live.

Hierarchy nodes consist of a flexible arrangement of links to items, which are used to organize and hierarchically represent the items found in volumes. Unlike volumes and volume groups, hierarchy nodes cannot be created, edited or removed. Instead, they are automatically generated in accordance with rules specified in [Cinemo MM Configuration](#).

Volumes and hierarchy nodes are organized into *volume groups*. Volume groups can be created as children of other volume groups. The root container is a volume group.

2.3.1 Volumes

All media devices are represented by Cinemo MM as volumes. A volume can represent any device with media content, e.g. USB devices, CDs, hard disk folders. However, this is not the only use for volumes – an application

may create volumes which do not correspond to any particular device, and which simply act as searchable repositories of persistent or non-persistent metadata. This type of volume is called a virtual volume.

Volumes can be *mounted* or *dismounted*. This is similar in meaning but should not be confused with operating system terminology. When a volume is mounted, the contents of the volume are active and searchable. When a volume is dismounted, the contents of the volume are inactive, and will not appear in browse or search results. Mounted volumes typically correspond to devices which are currently connected and whose media content is accessible; dismounted volumes corresponds to devices which have been connected in the past, but are currently disconnected. There may be many volumes, but only a few of these volumes will currently be mounted, corresponding to the devices which are currently connected. A dismounted volume has no children, and will consume no system resources – that is, RAM, threads, open file handles – apart from the storage required for its persistent database.

There are two types of volume, *indexed* and *virtual*.

An indexed volume represents a media device which is indexed by Cinemo, in other words, Cinemo is responsible for populating the volume container with files, folders and tracks and associated metadata from the device, and ensuring the volume is synchronized with the device as the media content changes over time. Indexed volumes are *persistent*, meaning that metadata extracted from media content on the device is stored locally in the Cinemo MM database. When the volume is mounted, Cinemo MM will ensure the contents of the persistent database are synchronized with the media content on the device.

A virtual volume, on the other hand, is managed by the application. The application is responsible for populating the virtual volume with files, folders, tracks and metadata. Cinemo MM does not restrict the class of object or metadata which can be written to a virtual volume – a virtual volume can be used by the application to represent anything at all. Virtual volumes can be *persistent* or *non-persistent*. When a persistent virtual volume is mounted, Cinemo MM will restore the contents of the volume to exactly the state in which the application last left it. The contents of a non-persistent virtual volume, on the other hand, will be lost when the volume is dismounted.

Indexed volumes can be created with [ICinemoMM::AddIndexedVolume\(\)](#). Virtual volumes can be created with [ICinemoMM::AddVirtualVolume\(\)](#). All volumes can be deleted by [ICinemoMM::RemoveNode\(\)](#). When a volume is deleted, all resources associated with the volume, including its persistent database, are freed.

Cinemo MM can also be configured to watch for USB device insertion and removal, and automatically create, mount and dismount indexed volumes for all supported devices. The list of currently supported devices can be obtained from your Cinemo representative. This may be convenient for an application, but the application designer should ensure that this is acceptable on the target platform. If this feature is enabled, Cinemo MM may inadvertently mount content not intended for indexing – for example, system firmware update discs.

2.3.2 Volume Groups

Volume groups are used to organize volumes into groups. All volumes, indexed or virtual, can only be created as children of a volume group. Likewise, volume groups can only be created as children of another volume group. The Cinemo MM root container is a volume group. There is no limit to the depth of a hierarchy of volume groups.

When an indexed or persistent virtual volume is created as a child of a volume group, the volume group and its parents up to the root container will also be persistent, and will be re-created each time Cinemo MM is started, as long as the persistent volume exists. Volume groups which contain no persistent descendants are also non-persistent, and will disappear when Cinemo MM is restarted.

[Cinemo MM Configuration](#) can be used to define an initial hierarchy of volume groups which is constructed when Cinemo MM is started. It is possible to add and remove volume groups with [ICinemoMM::AddVolumeGroup\(\)](#) and [ICinemoMM::RemoveNode\(\)](#).

Volume groups can be operated on in the same way as volumes. For example, it is possible to mount and dismount a volume group. Dismounting a volume group will cause all its child volumes to be recursively dismounted in a single atomic operation. When a volume group is removed, all its descendant volumes will be removed and their persistent resources freed.

2.3.3 Hierarchies

Hierarchy nodes are automatically generated by Cinemo MM in accordance with the rules specified in [Cinemo MM Configuration](#). Hierarchy nodes consist of a flexible arrangement of *links to items*, and are used to organize and hierarchically represent the content found in volumes.

Hierarchy nodes can only be generated as children of volume groups. Each volume group can specify its own hierarchy, which must be defined as a set of hierarchy rules in the MM configuration file. The configuration file may define multiple hierarchies, each identified by a unique hierarchy name, which must be specified when creating the volume group. The hierarchy for a volume group will consist only of links to items in the volumes descended from that volume group. It will not see items in volumes which are not descended from the volume group.

Hierarchy rules can be used to organize the content of a volume group by album, artist, genre, or any other application-specific metadata which Cinemo MM has been configured to recognize. It is important to note that Cinemo MM does not interpret any metadata which it has been configured to recognise. Metadata associated with items, for example, *album* and *artist*, are simply regarded as opaque fields which may be referred to in query expressions, sort criteria, and the rules for building hierarchy nodes. Cinemo MM knows nothing about albums or artists.

2.4 Indexed Volumes

Cinemo MM incorporates a fast content indexing engine, tested on storage devices containing hundreds of thousands of multimedia files. A multi-pass incremental indexing approach ensures that media content on a device is available for browse and search within fractions of a second, long before the full indexing process is complete. The Cinemo MM indexing engine is built on top of standard Cinemo interfaces, and is therefore capable of recognizing all media formats supported by Cinemo Media Engine. For a list of supported formats, please contact your Cinemo representative.

Cinemo MM indexing process is started whenever an indexed volume is mounted. If the volume has previously been mounted, then its persistent database will already contain some or all of the indexed content, in which case, Cinemo will only index the *delta* – that is, the difference between its persistent copy of the device, and the current state of the device.

Indexed volumes maintain their own background indexing threads, which run in parallel with other indexed volumes, and which do not lock the database while running. There is no requirement for an application to monitor the indexing status of a volume, unless it wishes to do so for UI purposes, for example, to render a progress bar. All Cinemo MM operations will continue to operate uninterrupted while indexing threads are running. If a volume is dismounted or removed, its indexing process will be automatically stopped.

Cinemo MM indexing process consists of multiple passes over the storage device, as described in the following sections. In most cases, if the device has previously been indexed, the delta from its persistent database will be zero, in which case indexing will finish when pass 1 is complete.

2.4.1 Pass 1 – Filenames

The entire filesystem of the storage device will be traversed recursively in depth-first order, and the name of each file and folder will be compared with the persistent database. Each file and folder will be added to the volume container *as they are discovered*. This ensures that the mounted volume consists only of files and folders which actually exist. Hidden files and folders will be ignored.

If files and folders are discovered which have not been indexed before, or which appear to have changed since the last time they were indexed, they will be queued for metadata extraction in Pass 2. If previously indexed files and folders are missing on the storage device, they will be deleted from the database, thus reclaiming database storage space. Otherwise, an exact match has occurred, and no further action is required.

It normally takes less than a second to complete the filename pass. This does, however, depend on the speed of the storage device, the number of files and folders, and the speed of the operating system filesystem routines. Note that no data has been read from individual files. If new files were discovered and added to the volume, the only metadata that will be known for those files is the file *name, size* and *modification time*. Cinemo MM can be configured to make an initial classification of the file – e.g. as music, video, or photo – depending on the file extension, which means that links to the file will appear in the correct location of the hierarchy depending on hierarchy building rules. However the class of file will not be accurately known until metadata extraction – file extensions are often erroneous.

When Pass 1 is complete, the persistent database will contain an up-to-date record of each file and folder on the storage device, except for any metadata which is still queued for extraction. This ensures that file and folder Object IDs are fully persistent next time the volume is mounted, even if the volume is dismounted before indexing is complete. The persistent database also maintains a *syncflag* for each file, indicating which indexing pass has been completed for that file. This ensures that when indexing is interrupted, it can be resumed next time the volume is mounted.

2.4.2 Pass 2 – Metadata Extraction

All files which were queued for metadata extraction in Pass 1 will be processed in Pass 2. If the volume has been previously indexed and no changes were discovered, then the metadata extraction queue will be empty and there will be nothing to do here.

The metadata extraction queue is normally processed in first-in-first out order, that is, in the order in which files were discovered. However, a *prioritization algorithm* ensures that tracks which currently appear in Cinemo MM browse and search results, and which are also queued for metadata extraction, are bumped to the head of the queue. Cinemo MM [Change Notifications](#) ensure that newly extracted metadata is notified immediately to the [ICinemoVFS](#) and [ICinemoPlaylist](#) interfaces in which the corresponding tracks appear, delivering the best possible browsing experience to the end user.

Metadata extracted from files will be separated into the fields which Cinemo MM has been configured to recognize in [Cinemo MM Configuration](#), and written to persistent storage. All unrecognised metadata will be discarded. Hierarchy nodes for the file will be reconstructed based on the new metadata and the hierarchy rules defined in Cinemo MM Configuration, for each parent volume group.

Cover Art and other types of EXIF thumbnail will also be extracted during this pass, since they are readily available in the extracted metadata. However, video and image thumbnails, which require high CPU resources to decode, will not be generated at this point. All video and image files for which no cover art exists will be queued for thumbnail generation in Pass 3.

2.4.3 Pass 3 – Thumbnails

All files which were queued for thumbnail generation will be processed here in Pass 3. Thumbnail generation is a CPU intensive process and this pass usually takes the longest to complete. Image files will be decoded, resized, and persistent thumbnails will be stored at a configurable resolution and image format (JPEG or PNG). Video files will be scanned using advanced algorithms for the most appropriate key frame.

2.4.4 Indexing Status

An application may wish to monitor the indexing status of a volume. This can be achieved by examining the [ICinemoMetapool](#) returned by opening the volume with [ICinemoVFS](#) or [ICinemoPlaylist](#).

If a volume group only contains one descendent volume, then the attributes of the volume are also returned as metadata for the volume group – the status of the volume group essentially *emulates* the volume. If an application wishes to monitor the status of multiple volumes, or multiple volume groups, it can open the parent container and read the metadata for each child.

An example user interface for monitoring volume status:

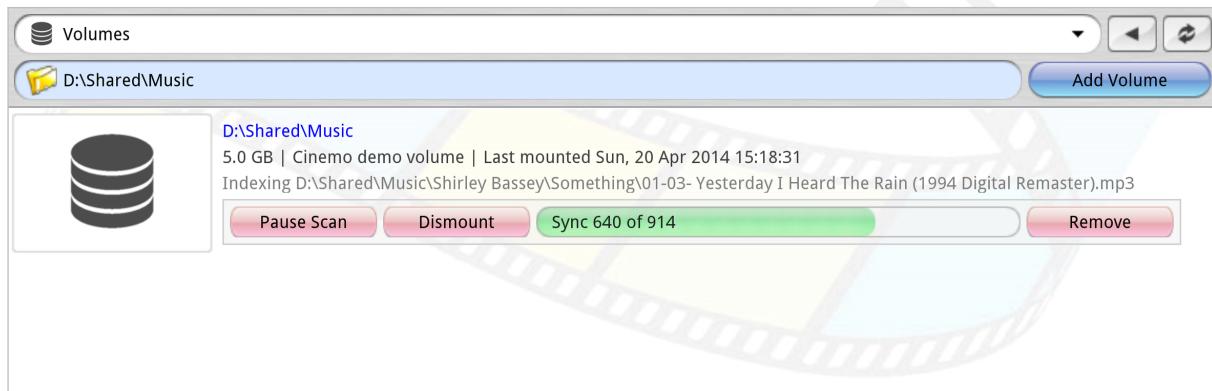


Figure 2.3: Volume status in Cinemo Demo App

The indexing status of each volume, including additional attributes such as its *mounted* or *paused* state, is indicated by volume-specific [Fields](#). These are as follows:

- [CINEMO_METANAME_VFS_UPNP_VOLUME_MOUNTPATH](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_UUID](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_TYPE](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_STATUS](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_STATUS_FLAGS](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_ERROR](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_MOUNTED](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_INDEXER_STATUS](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_INDEXER_PAUSED](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_PROGRESS](#)
- [CINEMO_METANAME_VFS_UPNP_VOLUME_PROGRESS_LENGTH](#)

2.4.5 Indexing Errors

The indexing of a volume may abort with an error code. Currently there are only two errors which can cause indexing to stop. These are [CinemoErrorDiscEjected](#) and [CinemoErrorNotEnoughSpace](#). In all cases, the error status of a volume can be determined by examining [CINEMO_METANAME_VFS_UPNP_VOLUME_ERROR](#).

[CinemoErrorDiscEjected](#) is returned when the [mount path](#) of an indexed volume does not appear to exist. If, for example, a USB mass storage device is forcibly removed during indexing, the first thing that will happen is that Cinemo MM will fail to open files on the device. The mount path is then checked to see if it is valid. If the mount path is no longer valid, then indexing will stop with [CinemoErrorDiscEjected](#). If, on the other hand, the mount path appears to be valid, it can be assumed that the file was deleted sometime between Indexing Pass 1 and the current time. In this case, the file will be removed from the volume hierarchy and from persistent storage, and indexing will continue with the remaining files.

[CinemoErrorNotEnoughSpace](#) is returned when either the RAM quota or the storage quota would be exceeded. In the case of exceeding the RAM quota, internal UPnP nodes cannot be allocated. In the case of exceeding the storage quota, persistent database files cannot be written. Indexing will stop in both cases. Note, the quotas are *never* exceeded. Indexing will stop to prevent this occurring.

2.5 Virtual Volumes

A virtual volume can be thought of as the root node of a filesystem which is writable by an application. A virtual volume contains virtual *files* and virtual *folders*. The files are items and the folders are containers. A virtual volume structure is therefore quite simple and can be represented as follows:

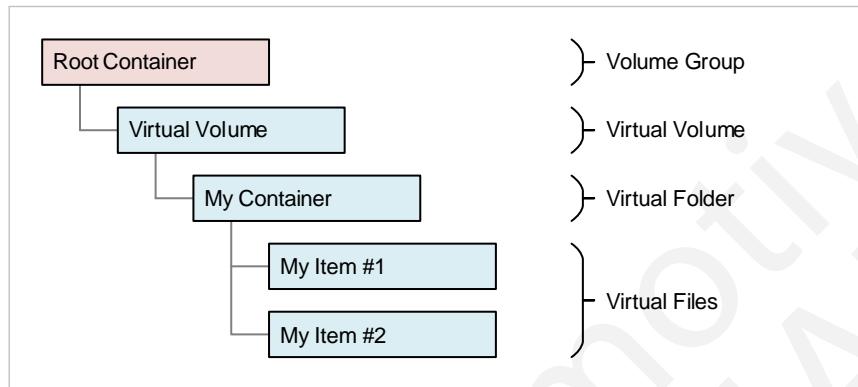


Figure 2.4: A simple virtual volume

It is worth noting that Cinemo MM treats indexed volumes and virtual volumes as the same thing. The internal implementation is the same. The only difference is, Cinemo MM will automatically populate the contents of an indexed volume. An indexed volume can therefore be thought of as a *persistent virtual volume* with an indexer behind it – all operations that Cinemo MM performs internally with indexed volumes could also be implemented by an application using virtual volumes.

The following methods are used to create and write to virtual volumes:

- [ICinemoMM::AddVirtualVolume\(\)](#)
- [ICinemoMM::AddVirtualFile\(\)](#)
- [ICinemoMM::AddVirtualFolder\(\)](#)
- [ICinemoMM::EditNode\(\)](#)
- [ICinemoMM::RemoveNode\(\)](#)

A virtual volume cannot be written unless it is *mounted*. All operations which operate on volume contents, such as [ICinemoMM::AddVirtualFile\(\)](#) and [ICinemoMM::EditNode\(\)](#), will fail if the volume is dismounted. A dismounted volume consumes no system resources: it has no thread to execute commands, and no internal hierarchy.

Virtual volumes, unlike indexed volumes, can be *persistent* or *non-persistent*. When a persistent virtual volume is mounted, Cinemo MM will restore the contents of the volume to exactly the state in which the application left it. The contents of a non-persistent virtual volume, on the other hand, will be lost when the volume is dismounted.

Virtual volumes, like indexed volumes, contain *items*, which are represented by virtual files. Hierarchy nodes will be automatically generated by Cinemo MM based on the content of metadata written to virtual files in accordance with hierarchy rules specified in [Cinemo MM Configuration](#), in the same way as for indexed volumes. Hierarchy nodes will contain *links to items* in the virtual volume. Whenever the metadata of a virtual file is

changed – for example, by [ICinemoMM::EditNode\(\)](#) – the hierarchy nodes will be updated to reflect the changed content. This is performed automatically by Cinemo MM virtual volume threads.

The following sections describe some typical use-cases for virtual volumes.

2.5.1 Unsupported Devices

It may be that an application has access to some media device, for example a USB or network device, for which there is no built-in support by Cinemo MM. In this case, an indexed volume cannot be used to represent the device, but instead, the application can create a virtual volume, and populate the virtual volume itself with the media content of the device.

There is one caveat: Cinemo MM must be able to *serve* the media content of the device, to local and networked Cinemo or third-party UPnP clients. This can be achieved with [ICinemoMM::CreateVirtualFileServer\(\)](#). The application will receive a callback whenever Cinemo MM is required to serve the contents of a virtual file, and must return from the callback a valid [ICinemoFile](#) object which can be used to read the file contents. The application must also return a valid *mime type*, which is required by Cinemo MM to serve files over HTTP.

An application may choose to create a persistent or non-persistent virtual volume for indexing its own devices. If a persistent virtual volume is selected, then the application is responsible for updating the contents of the volume to reflect changes in the device, each time the volume is mounted. If, on the other hand, a non-persistent volume is selected, then the application must re-build the entire contents of the volume each time it is mounted. This decision will depend on the application design and the properties of the device itself – for example, are the contents of the device likely to change, and how fast is the device to index.

2.5.2 Inter-Process Communication

A virtual volume can be accessed either locally or remotely. An application may edit the contents of a virtual volume from a remote location by calling [ICinemoMM::ConnectToServer\(\)](#), after which it will have full access to all volume functions such as [ICinemoMM::AddVirtualFile\(\)](#) and [ICinemoMM::EditNode\(\)](#).

All changes to a virtual volume are immediately notified to remote UPnP clients by means of [Change Notifications](#), so virtual volumes can be used as kind of inter-process communication mechanism.

For example, on one machine, [ICinemoMM](#) is used to write to the volume, and on another, [ICinemoVFS](#) or [ICinemoPlaylist](#) is used to watch for changes. The Cinemo MM server itself may be physically located on a third machine. Any application-specific metadata defines as custom metadata fields in [Cinemo MM Configuration](#) can be transmitted from one machine to another in this way. There are an unlimited number of potential application-specific use cases for this feature – for example, an application may wish to transmit its currently playing *playlist* to remote clients.

2.5.3 Arbitrary Metadata

A virtual volume may be used not only to represent the contents of a media device, but to represent any metadata at all which an application may wish to hierarchically arrange in a volume structure.

There are some benefits to storing metadata in a virtual volume which an application may wish to take advantage of. A virtual volume can be used as a *persistent repository* for metadata – as long as the volume is mounted, an application will have local or remote read and write access to the contents of the volume, which will persist between system shutdown and restart. In addition, custom hierarchy rules can be defined to organize the content of a virtual volume for browsing and searching in a manner appropriate to the content. The following example demonstrates a hierarchy customized for the display of contact information:

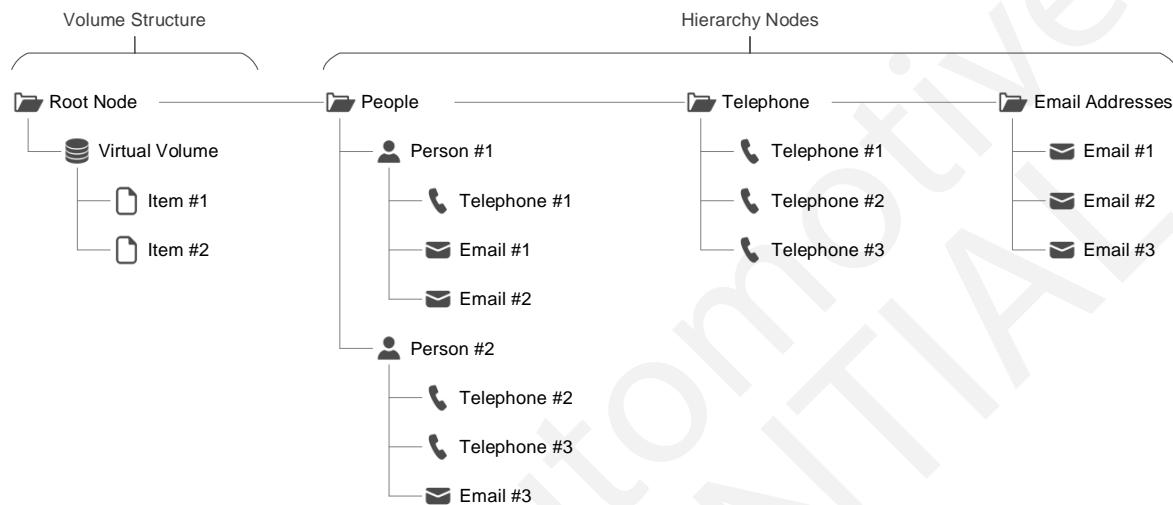


Figure 2.5: A customized virtual volume hierarchy

In the above example, the only objects directly managed by the application are Item #1 and Item #2, created with [ICinemoMM::AddVirtualFile\(\)](#). Each item contains application-specific metadata consisting of variable counts of telephone numbers and email addresses, as described in [Indexed Fields](#). The hierarchy nodes for organizing this metadata were constructed by Cinemo MM in accordance with application-specific hierarchy rules defined in [Cinemo MM Configuration](#).

An application can choose how to store metadata within virtual files. In the above example, additional metadata fields were defined for telephone numbers and email addresses, with the possibility to assign multiple fields to each item. It is also possible to construct virtual folders and separate the metadata between files in multiple folders. The best way to represent an application-specific dataset is a decision for the application designer. Please contact your Cinemo representative for further advice on this matter.

2.6 Content Recognition Services

Cinemo MM exposes a set of APIs which support integration with third-party content recognition services such as Gracenote™ and Nuance™.

Note that Cinemo MM **does not integrate support** for these services, and does not link to or require access to their corresponding SDKs and libraries – instead, [ICinemoMM](#) exposes generic methods and callbacks which enable an application to integrate support for all known recognition services. Cinemo MM sample code is provided as a guide, which may however require linking against e.g. GNSDK.

The relationship between Cinemo MM and content recognition services on a target system:

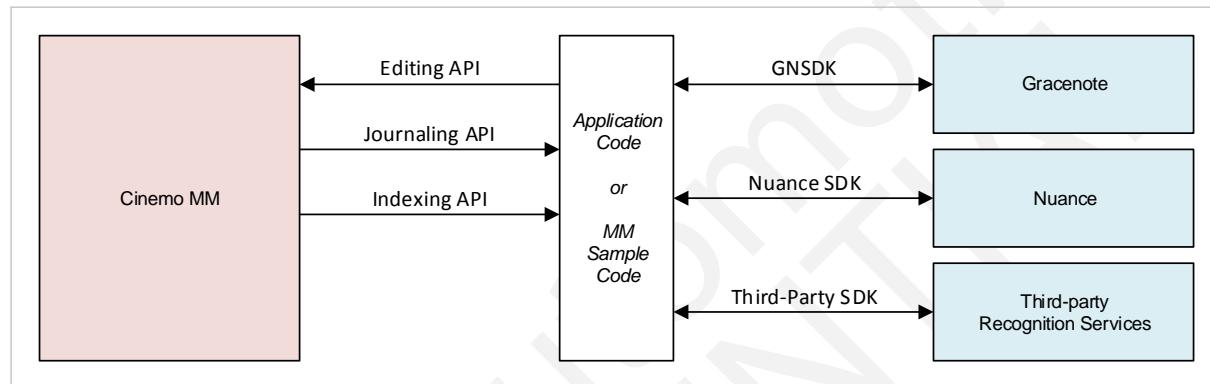


Figure 2.6: Content Recognition Services

Cinemo MM exposes the following APIs:

- An *Editing API* which allows the metadata fields of items in indexed and virtual volumes to be changed, enabling the writing of improved metadata from content recognition services, plus any additional metadata required to keep track of, or maintain state of these services.
- A *Journaling API* which allows an application to keep track of, and know at any point in time, the entire Cinemo MM database contents, enabling the synchronization of Cinemo MM database with a third party database.
- An *Indexing API* which allows an application to keep track of, and know at any point in time, the indexing state of any item within an indexed volume, enabling an application to know *when* to make use of content recognition services, and to request additional indexing passes if required.

These APIs are described in detail in the following sections.

2.6.1 Editing API

After connecting to either a local or a remote Cinemo MM server, the editing API provides functionality to modify all existing nodes. Using [ICinemoMM::Edit\(\)](#), metadata can be modified or new metadata be added to existing nodes. Only nodes on mounted volumes may be edited, for nodes on unmounted volumes *CinemoErrorNoSuchObject* is returned.

Changes will be distributed to all clients connected to the MM server using dynamic updates.

2.6.2 Journaling API

Using the Journaling API, it is possible to receive notifications if the state of the Cinemo MM database changes. The Journaling API provides the means to synchronize an external database to the MM database at very little effort.

There are two kinds of notifications available, which need to be enabled separately. The first kind are related to volumes, the second kind to individual nodes on a specific volume.

Cinemo MM does provide journaling data for volume and node operations but not for the metadata themselves. It is not possible to restore metadata for specific revisions, only the latest set of metadata is stored in the database and available for browsing and searching.

Volume events

With [ICinemoMM::StartVolumeEvents\(\)](#) it is possible to subscribe to events related to changes in the volume structure of the MM database. Each callback includes a *CinemoMMVolumeEvent* structure, which contains the actual volume events. A volume event, in turn, contains the volume identifier, the timestamp of the event and an event code, which can be one of *Existing, Add, Remove, Mount, Dismount, or Status*.

The callback also receives a *CinemoError* code, which needs to be checked every time. If the error code is *CinemoNoError* the callback contains valid data. In all other cases an error has occurred, the processing of volume events is aborted and no further volume events will be reported.

To disable volume events, [ICinemoMM::StopVolumeEvents\(\)](#) can be called.

Node events

The [ICinemoMM::StartNodeEvents\(\)](#) method can be used to register a callback function to receive node events for a specific volume. In order to receive node events for multiple volumes, individual callbacks for each volume need to be registered. [ICinemoMM::StartNodeEvents\(\)](#) accepts a revision identifier, which can be NULL. All node events, starting from the specified revision, will then be reported via the callback function. Each callback includes a revision identifier, the volume identifier of the node and a *CinemoMMNodeEvent* structure, which contains the actual node events.

The *CinemoMMNodeEvent* structure includes the following information:

- Node Id for the respective node
- Event code (either *Existing, Add, Remove, Change, or Status*)

- Fields of the metadata which have changed
- Optional metapool containing the most recent metadata on *Existing*, *Add*, or *Change* events

The callback also receives a *CinemoError* code, which needs to be checked every time. If the error code is *CinemoNoError* the callback contains valid data. In all other cases an error has occurred, the processing of node events is aborted and no further node events will be reported.

To disable node events, [ICinemoMM::StopNodeEvents\(\)](#) can be called.

Revision identifiers

It is important to note that there is no global revision identifier for the Cinemo MM database. There is a separate revision identifier for node events of each volume.

The structure of revision identifiers may change in the future and should not be assumed to be simple integer numbers.

2.6.3 Indexing API

The Indexing API can be used to retrieve notifications about the indexing process of nodes in Cinemo MM. It allows customization of the indexing process with additional indexing passes, e.g. adding an external content recognition service.

Indexing Events

To start receiving indexing events for a specific volume, [ICinemoMM::StartIndexingEvents\(\)](#) needs to be called with a volume id and a callback function. For all items in this volume, which are indexed for the first time, the callback function will be called with a volume id and an array of *CinemoMMIndexingEvents*.

Each indexing event contains the following information:

- The node id of the respective node
- The indexing pass id
- Indexing event flags
- The metapool for this item

Each indexing event contains the entire metapool for the indexed node. In order to add or modify the pool for this node, [ICinemoMM::EditNode\(\)](#) has to be called.

To disable node events, [ICinemoMM::StopIndexingEvents\(\)](#) can be called.

Understanding the indexing pass id

An integral part of the Indexing API is the `indexing_pass_id`. It determines which items do trigger an indexing event and what happens if the volume is mounted at a later time.

The `indexing_pass_id` is available in the metapool of each node with the metaname `CINEMO_METANAME_VFS_UPNP_INDEXING_PASS`. It can be edited like all other metadata by calling [ICinemoMM::EditNode\(\)](#) with the proper arguments.

If the `indexing_pass_id` is set to `CINEMO_MM_INDEXING_PASS_COMPLETE`, it will no longer appear in any future indexing event, even when remounting the device.

All volume indexers in Cinemo MM set the value of `indexing_pass_id` of each node to 1 as soon as the indexing of that particular node is completed.

If indexing events are started, a callback is generated for all nodes where the `indexing_pass_id` is non-zero and not `CINEMO_MM_INDEXING_PASS_COMPLETE`. First all nodes where the `indexing_pass_id` equals 1 are processed. Once all of these nodes have been processed, all nodes with an `indexing_pass_id` of 2 will be queued for processing.

It is possible to requeue a node for a later stage of the indexing pass by changing the `CINEMO_METANAME_VFS_UPNP_INDEXING_PASS` of the respective node via the Editing API to a different value. Changing the value to `CINEMO_MM_INDEXING_PASS_COMPLETE` will remove it from the indexing event queue.

The event flags `CINEMO_MM_INDEXING_EVENT_FLAG_FIRST` and `FLAG_LAST` indicate the beginning and end of each burst of events. In particular, the first node in each indexing pass will have the `CINEMO_MM_INDEXING_EVENT_FLAG_FIRST` flag set, the last node will have the `CINEMO_MM_INDEXING_EVENT_FLAG_LAST` flag set.

2.6.4 Implications

The three aforementioned APIs – the Editing API, Journaling API, and Indexing API – are closely tied together. When using all of them together, it is necessary to fully understand all consequences of each function call.

- **Editing API**

Each call of [ICinemoMM::EditNode\(\)](#) will create a `CinemoMMNodeEvent` in the Journaling API. If the `CINEMO_METANAME_VFS_UPNP_INDEXING_PASS` is changed, it may also create a callback via the Indexing API.

- **Journaling API**

The Journaling API should only be used to keep in sync with the Cinemo MM database. It should never be used to modify the metadata of the nodes, because any metadata might be overwritten by an internal indexing pass of Cinemo MM.

- **Indexing API**

All metadata editing should be performed as a result of a `CinemoMMIndexingEvent`. This ensures data integrity with internal Cinemo MM indexing passes. If a node is edited as a result of a `CinemoMMIndexingEvent`, it will create a node event in the Journaling API. If the `CINEMO_METANAME_VFS_UPNP_INDEXING_PASS` is changed, it may also create another callback via the Indexing API during a later indexing pass.

2.7 Query Language

Cinemo MM incorporates a powerful expression parser enabling the real-time execution of complex queries on large datasets. In addition to fully supporting the UPnP query syntax, Cinemo MM supports syntax extensions which allow applications to query for any known attribute of the indexed media content – for example, textual data such as album or artist, or video resolution, audio bitrate, number of audio channels, container format, and so on.

Queries operate on fields. The UPnP query expression syntax is specified in [UPnP Content Directory Service 1.01](#) section 2.5.5.1. An example expression is provided here:

```
upnp:class derivedfrom "object.item.audioItem" and (
    dc:title contains "madonna" or
    upnp:artist contains "madonna" or
    upnp:album contains "madonna" or
    upnp:albumArtist contains "madonna") and @refID exists false
```

This expression will return all *audio items* whose *title*, *artist*, *album* or *albumArtist* metadata contains the text “Madonna”, and which are not *links*. This final step ensures that the same track is not returned more than once in the results, since a track may be referred to by one or more links, each of which match the search expression, and each of which exist at different locations within the hierarchy – for example, under ‘All Music’ and also under ‘Music/Artists/Madonna’.

It is worth noting that the UPnP query language, although powerful, is not SQL. A query expression is similar to a *regular expression* in the sense that it is used for *matching* objects, in the same way that a regular expression is used for matching strings. When used in a search, the query returns a boolean result for each object it is evaluated against. If the result is *true*, the object is returned in the search results. A query can only return objects **that exist in the hierarchy**. This is why media servers go to so much trouble building [Cinemo MM Hierarchies](#) – it is only during hierarchy building that new objects, such as *Artists* and *Albums*, can be created. Once objects are created, they can be returned in query expressions.

2.7.1 Indexed Fields

Queries can operate on all metadata fields which Cinemo MM has been configured to recognise. In addition, Cinemo MM extends the UPnP specification by supporting *indexed fields*. Cinemo MM can store multiple metadata of each field type, identified by the *index* parameter of [ICinemoMetapool](#) metadata attributes. This applies to audio, video, thumbnails, and all custom fields (which is most fields). It does not apply to certain built-in fields such as *@id* and *res@size* – an object can only have one ID and one size.

The UPnP query syntax is extended to support this. For example, *dc:title* refers to the first available *Title* metadata, and is therefore compatible with UPnP specification, but *dc:title[0]* and *dc:title[1]* refer specifically to *Title* metadata with the corresponding index.

Indexed fields may be used for storing metadata attributes such as chapter names, telephone numbers, or any other attribute an application may define in [Cinemo MM Configuration](#). The index of a field must be in the range [0 - 255], represented internally by an unsigned byte.

2.7.2 Operators

Cinemo MM supports relational operators in query expressions.

All operators defined in [UPnP Content Directory Service 1.01](#) are fully supported, and some additional operators are implemented as extensions to the UPnP specification.

Some operators supported for compliance with the UPnP specification are redundant, since they may be more compactly and efficiently expressed in terms of other operators. For example, the following expression pairs are logically equivalent to each other:

EXPRESSION	EQUIVALENT EXPRESSION
<code>@refID exists false</code>	<code>!@refID</code>
<code>dc:title doesnotcontain "mad"</code>	<code>!(dc:title contains "mad")</code>

Cinemo MM supports mathematical expressions on 64-bit integer values. For example, the following expressions are correctly evaluated:

EXPRESSION
<code>res@bitrate / 1000 >= 320</code>
<code>cinemo:videoCx * cinemo:videoCy <= 10000</code>

All supported operators are listed here in order of precedence, from highest to lowest:

OPERATOR	DESCRIPTION
<code>""</code>	Double quotation marks to encapsulate text strings. Inside text strings escaping is required for double quotation marks (\") and backslashes (\ \).
<code>(</code>	Left parenthesis
<code>)</code>	Right parenthesis
<code>regexreplace</code>	Replace substring matching a regular expression
<code>childcountofclass</code>	Number of children derived from <code><upnp:class></code>
<code>containsphonetic</code>	<code><text field> containsphonetic <text> <pattern></code>
<code>startswithphonetic</code>	<code><text field> starts with phonetic <text> <pattern></code>
<code>-</code>	Unary negation (when no left term)
<code>!</code>	Unary logical NOT
<code>~</code>	Unary bitwise NOT
<code>*</code>	Multiply
<code>/</code>	Divide
<code>%</code>	Modulo
<code>+</code>	Add

OPERATOR	DESCRIPTION
-	Subtract (when left term exists)
=	Equal
!=	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
contains	<text field> contains <text>
doesnotcontain	<text field> does not contain <text>
startswith	<text field> starts with <text>
regexmatch	Match regular expression
derivedfrom	<upnp:class> is derived from <upnp:class>
exists	<any field> exists [true,false]
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
&&	Logical AND
and	Logical AND
	Logical OR
or	Logical OR

2.7.3 Sorting

Cinemo MM incorporates a powerful server-side result sorting mechanism. In addition to fully supporting the UPnP sorting syntax, CinemoMM supports not only sorting based on all known attributes of media content but also allows to formulate complex sort expressions based on the MM query language.

A sorted result is achieved by appending the sort URL parameter to search or browse requests. The following URL will return a sorted search result.

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?
  pid=<id>&
  search=<query>&
  sort=<sort>
```

According to the UPnP specification, the <sort> parameter is a comma separated list of fields to be used for sorting in either ascending or descending order.

```
sort=+cinemo:order,+dc:title
```

CinemoMM extends the UPnP sorting mechanism in order to support a wider range of sorting tasks by allowing to specify query language expressions in the sort parameter. Using expressions in the sort parameter allows to prioritize specific items by properties that are not directly accessible as metadata fields.

```
sort=+$(upnp:class derivedfrom object.container ? 0 : 1),+dc:title
```

For text matching queries, CinemoMM supports sorting the result based on properties of the text match such as match position or match type. For matching items, the match properties are stored as result context variables. Result context variables are accessible using the query language. The following result context variables are available for sorting:

IDENTIFIER	OPERATORS	DESCRIPTION
\$matchpos	contains startsWith containsPhonetic startsWithPhonetic regexmatch	Position of first match in field
\$matchlen	contains startsWith containsPhonetic startsWithPhonetic regexmatch	Length of first match in field
\$matches	contains startsWith containsPhonetic startsWithPhonetic regexmatch	Number of matches in field
\$phonetic.matches.initial	containsPhonetic startsWithPhonetic	Number of initial phonetic matches
\$phonetic.matches.full	containsPhonetic startsWithPhonetic	Number of full phonetic matches
\$phonetic.matches.mixed	containsPhonetic startsWithPhonetic	Number of mixed initial and full phonetic matches
\$word.matches.exact	containsWord	Number of exact word matches
\$word.matches.partial	containsWord	Number of partial word matches

The following query sorts the result according to the text match position. Results having the same match position are sorted alphabetically.

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?  
pid=<id>&
```

```
search=dc:title contains Madonna&
sort=+$($matchpos),+dc:title
```

Cinemo Media Management also supports several specialized sorting algorithms. A specific algorithm can be selected by adding a selection character code before the field identifier:

```
upnp://<addr:port>/0/MediaServer/DeviceDesc.xml?
pid=<id>&
browse=BrowseDirectChildren&
sort=+{CODE}dc:title
```

The following table gives an overview of available algorithms by character code.

CODE	DESCRIPTION
\$	Sort by expression, see above.
#	Sort by unicode code point. Only applicable to text fields.
>	Sort recursive from root, breadth-first, prefer files before folders. Only applicable to text fields.
<	Sort recursive from root, depth-first, prefer files before folders. Only applicable to text fields.

2.8 Extended Text Matching

In order to support the most common text input and query methods, Cinemo MM extends the text matching capabilities of UPnP by means of additional matching operators.

2.8.1 Word Boundary Matching

In Cinemo MM, matching at boundaries of words is supported by means of the *containsWord* operator:

```
<field> containsWord <query>
```

The *<field>* parameter defines the metadata field to be used for matching.

The *<query>* parameter defines the query characters to be used for word boundary matching.

The word boundary identification can be controlled by the two options [CINEMO_OPTION_MM_ICU_MODE](#) and [CINEMO_OPTION_MM_ICU_SEARCH_LOCALE](#). If ICU search mode is used, the word boundary identification is based on the word boundary data file associated to the ICU search locale. If ICU search mode is not used, all ASCII symbols and punctuation marks are used for word boundary identification.

Cinemo MM allows to define the mode used for word boundary matching by means of the additional URL parameter *wmmode*. The *wmmode* can take one of the following values.

WMMODE	DESCRIPTION
partial	Respond partial and exact word matches (default)
exact	Respond only exact word matches

2.8.2 Phonetic Matching

Phonetic matching refers to the task of matching query characters in a phonetic representation of metadata. Such a phonetic representation could be e.g. Pinyin for simplified Chinese metadata or BoPoMoFo for traditional Chinese metadata.

In Cinemo MM, phonetic matching can be realized by means of the *containsPhonetic* and *startsWithPhonetic* operators. The phonetic operators perform matching based on two metadata fields:

- The original metadata field defined as *<field>* in the query.
- The associated phonetic field defined in the [field definition](#).

The phonetic field needs to be provided to Cinemo MM by the application using the [Indexing API](#). The application needs to convert the metadata to the desired phonetic representation (e.g. Pinyin) and store the phonetic representation with MM using the [Editing API](#).

The content of phonetic fields shall meet the following conventions:

- Syllables are identified by using the the \$ prefix.

- Non-converted characters are represented using an empty syllable.
- Trailing empty syllables can be omitted.

The following table provides examples on the phonetic field syntax.

WRITTEN	PHONETIC
很爱你	\$hen\$ai\$ni
你好 Hallo	\$ni\$hao
Hallo 你好	\$\$\$\$\$\$ni\$hao
Haiden Tseung	

The phonetic matching operator has the following syntax:

```
<field> containsPhonetic <query> <pattern>
```

The **<field>** parameter defines the metadata field to be used for matching.

The **<query>** parameter defines the query characters to be used for phonetic matching.

The **<pattern>** parameter defines how the **<query>** characters shall be matched. The pattern consists of one or multiple match types accompanied with the number of query characters to be used for the desired match type.

The following match types are available:

MATCHTYPE	DESCRIPTION
p	Match in phonetic representation
w	Match in written representation
m	Match in written representation or phonetic representation

For the last item, the number of query characters can be omitted when assembling the pattern.

The following expressions show exemplary patterns for the *containsPhonetic* operator.

- Match in phonetic representation only
`dc:title containsPhonetic "ha" p`
- Match in phonetic and written representation
`dc:title containsPhonetic "ha" m`
- Match substring of length 1 in written and remaining in phonetic representation:
`dc:title containsPhonetic "好ha" w1p`

Cinemo MM allows to define the mode used for phonetic matching by means of the additional URL parameter **pmmode**. The **pmmode** can take one of the following values.

PMMODE	DESCRIPTION
(F I)*	Any combination of full and initial phonetic matching (default)
F*I*	Full phonetic matching followed by initial phonetic matching
F*	Only full phonetic matching
I*	Only initial phonetic matching

2.9 Fields

[Cinemo MM Configuration](#) defines a number of fixed fields which must not be changed by an application, followed by a set of custom fields which can be freely modified, removed, or appended to. Fixed fields have special meanings as described in the following sections. Custom fields are simply regarded as opaque metadata and are not interpreted by Cinemo MM.

All fields for all item and container objects within indexed and virtual volumes, including the volume itself, can be written to or modified by [ICinemoMM::EditNode\(\)](#). All fields for hierarchy nodes, on the other hand, are automatically assigned in accordance with hierarchy rules which may be defined by an application in [Cinemo MM Configuration](#), and cannot be directly edited.

2.9.1 @id

2.9.2 @parentID

2.9.3 @refID

2.9.4 @volumeID

FIELD	TYPE	METANAME
@id	text	VFS_UPNP_ID
@parentID	text	VFS_UPNP_PARENTID
@refID	text	VFS_UPNP_REFID
@volumeID	text	VFS_UPNP_VOLUMEID

The `@id` and `@parentID` fields are exposed by all item and container objects, since all objects have a unique ID. In the case of the root container, `@id` is “0” and `@parentID` is “-1”, in accordance with UPnP specification.

The `@refID` field is exposed by links to items in hierarchy nodes. This is because the `@id` of a link is not equal to the `@id` of the referenced item, which can instead be determined by `@refID`, in accordance with UPnP specification. There may be multiple links to the same item in multiple locations within hierarchy nodes, all with the same `@refID`. When searching at the root node, it is often desired to exclude links from the results, which can be achieved with the query `!@refID` or `@refID exists false`.

The `@volumeID` field is exposed by items and containers within a volume, and identifies the parent volume container. This is a convenient shortcut: the parent volume of a given item or container could also be determined by following `@parentID` to the parent volume.

2.9.5 @childCount

2.9.6 @searchable

2.9.7 @restricted

FIELD	TYPE	METANAME
@childCount	ui32	VFS_UPNP_CHILDCOUNT
@searchable	ui32	VFS_UPNP_SEARCHABLE
@restricted	ui32	VFS_UPNP_RESTRICTED

These fields are exposed by containers, not items. `@childCount` specifies the count of children of the container, `@searchable` specifies if the container is searchable, and `@restricted` specifies if the container is restricted.

A searchable container is capable of executing query expressions – by default all Cinemo MM containers are searchable, however this may not be the case for third-party UPnP media servers.

A restricted container is non-writeable, which means its attributes cannot be changed with [ICinemoMM::EditNode\(\)](#), and its children cannot be added, removed or edited by any of [ICinemoMM::AddVolumeGroup\(\)](#), [ICinemoMM::AddIndexedVolume\(\)](#), [ICinemoMM::AddVirtualVolume\(\)](#), [ICinemoMM::AddVirtualFile\(\)](#), [ICinemoMM::AddVirtualFolder\(\)](#) or [ICinemoMM::RemoveNode\(\)](#) methods. All hierarchy nodes are restricted, whereas all volume groups, volumes and their children are unrestricted.

2.9.8 upnp:class

2.9.9 upnp:searchClass

FIELD	TYPE	METANAME
upnp:class	text	VFS_UPNP_CLASS
upnp:searchClass	text	VFS_UPNP_SEARCHCLASS

The `@upnp:class` field is exposed by all objects, since all objects must be derived from a class. The class defines, for example, whether or not an object is an item or container.

The `@upnp:searchClass` field is only exposed by containers, and specifies the class of objects that a container is allowed to return from a search query.

If `@upnp:searchClass` is specified for a container, then care should be taken when executing searches, that the search query does not exclude the class of objects a container is allowed to return. For example, searching for “`upnp:class derivedfrom object.item`” on a container with search class `object.container.album.musicAlbum` will yield no results, since the query expression conflicts with the search class.

If the `@upnp:searchClass` field is not specified, then the container is allowed to return any object from a search, in accordance with UPnP specification.

2.9.10 cinemo:order

FIELD	TYPE	METANAME
cinemo:order	ui32	VFS_UPNP_ORDER

This field is exposed by all items and containers, and is intended for use in specifying sort criteria for browse or search results. For example, an application may wish to list folders before files in browse results. This can be achieved by assigning `cinemo:order=1` to all folders, and `cinemo:order=2` to all files. If the sort criteria is then specified as “`+cinemo:order,+dc:title`”, the results will be sorted with all folders in alphabetical order, followed by all files in alphabetical order.

There may be other ways to achieve the same result, and the `cinemo:order` field may be used by an application for other purposes. For example, an application could define its own custom fields, and sort by these instead.

The *cinemo:order* field simply exposes an efficient 32-bit value which is present for all items and containers, which can be used for anything at all, and which can be quickly accessed in query expressions and sort criteria without requiring the additional memory and CPU overhead of custom metadata fields.

2.9.11 *cinemo:updateID*

FIELD	TYPE	METANAME
<i>cinemo:updateID</i>	ui32	VFS_UPNP_UPDATEID

This field is only exposed by containers. It is incremented by Cinemo MM each time the contents of the container are modified – that is, if an immediate child is added or removed, or the metadata for an immediate child is changed. An application should not interpret or do anything with this field. It is used internally by [ICinemoVFS](#) and [ICinemoPlaylist](#) to determine if the contents of the currently browsing container have changed, and to deliver [Cinemo MM Change Notifications](#) if required.

2.9.12 *cinemo:name*

FIELD	TYPE	METANAME
<i>cinemo:name</i>	text	VFS_NAME

This field is exposed by all items and containers. All objects must have a valid name.

Note, the name of an object is not equal to its *dc:title* – for example, all files have names, which are typically not the same as the *dc:title* field which is extracted from metadata during Indexing Pass 2. Cinemo MM requires that all objects have a name, whereas the *dc:title* is simply regarded as optional metadata which may even be excluded from [Cinemo MM Configuration](#).

Since most third-party UPnP clients expect to receive *dc:title*, not *cinemo:name*, the default configuration defines *dc:title* with *cinemo:name* as a fallback – in other words, if *dc:title* metadata is not present for an object, then *dc:title* will be returned with the value of *cinemo:name* field instead, which is always present.

2.9.13 *res*

FIELD	TYPE	METANAME
<i>res</i>	text	VFS_PATH

The *res* field is the object's path. This field is exposed by all items and containers, since all objects must have a valid path. The path for an item is the playback URL – that is, the URL which [ICinemoPlayer](#) uses to access the media content of the item, usually an HTTP link to the item served by the media server. The path for a container is the browse URL – that is, the URL that would be passed to [ICinemoVFS](#) or [ICinemoPlaylist](#) to browse the immediate children of the container.

The path for any item within an indexed or virtual volume may be edited by an application by means of [ICinemoMM::EditNode\(\)](#). In this case, the edited path for an item will be returned verbatim in browse or search results. Care should be taken to ensure that the path is accessible to all UPnP clients which may browse or

search the media server, which may be local or remote. For example, paths which directly reference the local filesystem will not be accessible to remote UPnP clients.

2.9.14 **cinemo:type**

FIELD	TYPE	METANAME
cinemo:type	ui32	VFS_TYPE

This field is exposed by all items and containers, and corresponds to the [CINEMO_VFS_TYPE](#) of the object, as would be returned by opening the object with [ICinemoVFS](#). Checking this field is the easiest way to see if an object is an item or a container.

2.9.15 **cinemo:time**

FIELD	TYPE	METANAME
cinemo:time	ui32	VFS_TIME

This field is exposed by all volumes, and by all items within volumes. In the case of volumes, it indicates the time, measured in seconds elapsed since 1 Jan 1970, when the volume was last mounted. The current system time will be assigned when a volume is mounted. In the case of items within volumes, it indicates the last modification time of a file, in units of seconds since 1 Jan 1970. It is used by indexed volumes to determine if the contents of a file have changed since the last time the file was indexed. In the case of items within virtual volumes, this field may be used by an application for its own purposes.

2.9.16 **res@size**

FIELD	TYPE	METANAME
res@size	ui64	VFS_SIZE

This field is exposed by all volumes, and by all items within volumes. In the case of volumes, it indicates the total size in bytes of all items descended from a volume – not the database size of the volume. In the case of items within volumes, it represents the size of the corresponding file, in bytes, and must be accurately set in order for Cinemo MM to serve the file. In the case of items within indexed volumes, this field is automatically assigned when the file is indexed. In the case of items within virtual volumes, it is the application's responsibility to correctly assign this field.

2.9.17 **upnp:albumArtURI**

FIELD	TYPE	METANAME
upnp:albumArtURI	text	VFS_ICON_URL

This field is exposed by all items and containers which have a valid icon or album art. It corresponds to the download URL of the corresponding image data, typically an HTTP link.

The *upnp:albumArtURI* field for any object within an indexed or virtual volume may be edited by an application with [ICinemoMM::EditNode\(\)](#). In this case, the edited URL will be returned verbatim in browse or search results.

Care should be taken to ensure that the URL is accessible to all UPnP clients which may browse or search the media server, which may be local or remote. For example, URLs which directly reference the local filesystem will not be accessible to remote UPnP clients.

2.9.18 cinemo:audio

2.9.19 cinemo:video

FIELD	TYPE	METANAME
cinemo:audioType	text	N/A
cinemo:audioSubtype	text	N/A
res@sampleFrequency	ui32	N/A
res@nrAudioChannels	ui32	N/A
cinemo:audioChannelConfig	ui32	N/A
res@bitsPerSample	ui32	N/A
res@bitrate	ui32	N/A
cinemo:videoType	text	N/A
cinemo:videoSubtype	text	N/A
cinemo:videoCx	ui32	N/A
cinemo:videoCy	ui32	N/A
cinemo:videoInterlaced	ui32	N/A
cinemo:videoRatioX	ui32	N/A
cinemo:videoRatioY	ui32	N/A
cinemo:videoFrameDuration	ui32	N/A

These fields are maintained by items which have corresponding audio or video attributes. The type and subtype refer to the [CINEMO_MEDIATYPE](#) and [CINEMO_MEDIASUBTYPE](#) fields of the corresponding Cinemo media type, converted to text strings.

All of these fields can be used in query expressions, but are not returned in [ICinemoMetapool](#), since they are not generally required in this context and considered to be a waste of bandwidth. The [CINEMO_METANAME](#) is unspecified for this reason.

If an application wishes these values to be returned in the [ICinemoMetapool](#) for an object, the configuration file can be modified to enable this.

- 2.9.20 `cinemo:totalItems`**
2.9.21 `cinemo:totalAudioItems`
2.9.22 `cinemo:totalVideoItems`
2.9.23 `cinemo:totalImageItems`

FIELD	TYPE	METANAME
<code>cinemo:totalItems</code>	ui32	VFS_UPNP_FOLDER_TOTAL_ITEMS
<code>cinemo:totalAudioItems</code>	ui32	VFS_UPNP_FOLDER_TOTAL_AUDIO_ITEMS
<code>cinemo:totalVideoItems</code>	ui32	VFS_UPNP_FOLDER_TOTAL_VIDEO_ITEMS
<code>cinemo:totalImageItems</code>	ui32	VFS_UPNP_FOLDER_TOTAL_IMAGE_ITEMS

These fields are maintained by *volume groups*, *volumes* and *folders* contained within indexed and virtual volumes. They represent the total count of items of each audio, video and image class which are descended from the current location in the UPnP hierarchy. Since the root node is also a volume group, it can be used to determine the total count of items maintained by Cinemo MM.

2.9.24 `cinemo:volumeMountPath`

FIELD	TYPE	METANAME
<code>cinemo:volumeMountPath</code>	text	VFS_UPNP_VOLUME_MOUNTPATH

This field is maintained by volumes.

All indexed volumes require a mount path, specified as an argument to [ICinemoMM::AddIndexedVolume\(\)](#). It is possible to edit this field in a dismounted volume with [ICinemoMM::EditNode\(\)](#). The mount path of a currently mounted volume cannot be changed.

2.9.25 `cinemo:volumeUUID`

FIELD	TYPE	METANAME
<code>cinemo:volumeUUID</code>	text	VFS_UPNP_VOLUME_UUID

This field is maintained by volumes, and represents the unique ID of a volume, as specified in arguments to [ICinemoMM::AddIndexedVolume\(\)](#) or [ICinemoMM::AddVirtualVolume\(\)](#).

If this field is set, the value must be unique – a volume cannot have the same UUID as any other volume. Depending on Cinemo MM configuration options, this value may be automatically assigned when creating an indexed volume. This field can be edited with [ICinemoMM::EditNode\(\)](#).

2.9.26 `cinemo:volumeType`

FIELD	TYPE	METANAME
<code>cinemo:volumeType</code>	text	VFS_UPNP_VOLUME_TYPE

This field is maintained by volumes, and represents the volume type, which is an opaque string specified as arguments to [ICinemoMM::AddIndexedVolume\(\)](#) or [ICinemoMM::AddVirtualVolume\(\)](#).

Cinemo MM does not interpret the volume type – an application may use this field for its own purposes. This field can be edited with [ICinemoMM::EditNode\(\)](#).

2.9.27 **cinemo:volumeStatus**

FIELD	TYPE	METANAME
cinemo:volumeStatus	text	VFS_UPNP_VOLUME_STATUS

The volume status field is a non-persistent attribute of indexed volumes, indicating the current state of the volume in a human-readable manner. It may be set to one of the following values:

- “Paused”
- “Sync thumbnails”
- “Mounted”
- “Dismounting”
- “Dismounted”
- “Sync database”
- “Sync files”
- “Flushing”

These values should not be interpreted by an application, and may change in future versions.

2.9.28 **cinemo:volumeStatusFlags**

FIELD	TYPE	METANAME
cinemo:volumeStatusFlags	ui32	VFS_UPNP_VOLUME_STATUS_FLAGS

The volume status flags field is a non-persistent attribute of indexed volumes, indicating the current state of the volume in a machine-readable manner. The following flags may be set:

- CINEMO_MM_VOLUME_STATUS_FLAG_MOUNT
- CINEMO_MM_VOLUME_STATUS_FLAG_PAUSE
- CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_FASTPLAY
- CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_DATABASE
- CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_TRACKS
- CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_METADATA
- CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_THUMBS
- CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_DONE
- CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_EXTERNAL_DONE

The flag CINEMO_MM_VOLUME_STATUS_FLAG_MOUNT indicates, if the volume is mounted or not. It can be either 0 or 1, corresponding to the mounted state of the volume.

The flag CINEMO_MM_VOLUME_STATUS_FLAG_PAUSE indicates, if the indexing process of the volume has been paused. It can be either 0 or 1, corresponding to the paused state of the volume.

All CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER flags are set asynchronously during the initial indexing process and indicate that the respective pass of indexing has been accomplished.

The CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_EXTERNAL_DONE flag indicates that external indexing steps triggered by the [Indexing API](#) have been accomplished. If no external indexer is used, this flag will never be set.

For iAP2 devices, the device itself is responsible to provide the required information to set the CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER flags. It is not in Cinemo's hands whether all iAP2 devices do and will support this flag.

2.9.29 cinemo:volumeDismountStatus

FIELD	TYPE	METANAME
cinemo:volumeDismountStatus	ui32	VFS_UPNP_VOLUME_DISMOUNT_STATUS

The volume dismount status field is a non-persistent attribute of volumes, indicating the current state of dismounting the volume in a machine-readable manner. The following flags may be set:

- CINEMO_MM_VOLUME_DISMOUNT_STATUS_DISMOUNTING
- CINEMO_MM_VOLUME_DISMOUNT_STATUS_HIERARCHY
- CINEMO_MM_VOLUME_DISMOUNT_STATUS_DATABASE

The flag CINEMO_MM_VOLUME_DISMOUNT_STATUS_DISMOUNTING indicates, that the volume is currently performing an asynchronous dismount. Only if this flag is set, one of the following bits indicate the status of the dismounting sequence.

If CINEMO_MM_VOLUME_DISMOUNT_STATUS_HIERARCHY is set, all links to the hierarchy have been removed for the volume. If CINEMO_MM_VOLUME_DISMOUNT_STATUS_DATABASE is set, all outstanding database operations for items on the volume have been performed.

2.9.30 cinemo:volumeError

FIELD	TYPE	METANAME
cinemo:volumeError	text	VFS_UPNP_VOLUME_ERROR

The volume error field is a non-persistent attribute of indexed volumes, assigned by indexed volumes if indexing was aborted with an error. There are currently only two possible error codes which can cause the indexing process to stop:

- “CinemoErrorDiscEjected”
- “CinemoErrorNotEnoughSpace”

Additional error codes may be added in future versions.

2.9.31 **cinemo:volumeMounted**

FIELD	TYPE	METANAME
cinemo:volumeMounted	ui32	VFS_UPNP_VOLUME_MOUNTED

This is a non-persistent field indicating if the volume is mounted or not. It can be either 0 or 1, corresponding to the mounted state of the volume. Volumes can be mounted with [ICinemoMM::MountVolume\(\)](#) and dismounted with [ICinemoMM::DismountVolume\(\)](#).

2.9.32 **cinemo:volumeIndexerStatus**

FIELD	TYPE	METANAME
cinemo:volumeIndexerStatus	text	VFS_UPNP_VOLUME_INDEXER_STATUS

This is a non-persistent field indicating the current status of the indexing process. It can either be NULL, in which case the indexing process is stopped, or the full path name of the current file or folder which is currently being indexed.

2.9.33 **cinemo:volumeIndexerPaused**

FIELD	TYPE	METANAME
cinemo:volumeIndexerPaused	ui32	VFS_UPNP_VOLUME_INDEXER_PAUSED

This is a non-persistent field indicating the current pause state of the indexing process. It can be either 0 or 1, where 0 is running and 1 is paused. The state can be set with [ICinemoMM::PauseVolumeIndexer\(\)](#).

2.9.34 **cinemo:volumeProgress**

2.9.35 **cinemo:volumeProgressLength**

2.9.36 **cinemo:volumeProgressStatus**

FIELD	TYPE	METANAME
cinemo:volumeProgress	ui32	VFS_UPNP_VOLUME_PROGRESS
cinemo:volumeProgressLength	ui32	VFS_UPNP_VOLUME_PROGRESS_LENGTH
cinemo:volumeProgressStatus	ui32	VFS_UPNP_VOLUME_PROGRESS_STATUS

These are non-persistent fields indicating the current progress of the indexing process. The values should not be interpreted by an application. The ratio of *cinemo:volumeProgress* to *cinemo:volumeProgressLength* may be used to draw a progress bar. The value of *cinemo:volumeProgressStatus* indicates the current indexing phase by setting the appropriate *CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER_** flag.

2.10 Configuration

Cinemo MM is fully customizable with an XML configuration file. There is a default XML file built into the delivered binaries (`res://cinemo_mm.xml`), or an application may supply its own configuration file with `CINEMO_OPTION_MM_CONFIG_XML`. The configuration file enables customization of:

- UPnP classes
- UPnP fields
- UPnP browse hierarchy

Cinemo MM can be configured to store and hierarchically represent any kind of metadata which may be supplied by an application. Cinemo MM is optimized for use as a media server, since that is its original intended purpose, but it has been usefully configured as a searchable repository for metadata ranging from *contact information* in the form of an address book, to *EXIF attributes* for an image database.

Please contact your Cinemo representative if you have questions about additional use cases.

2.10.1 Classes

All classes recognized by Cinemo MM are defined in the *cinemo_mm_classes* section of the configuration XML. All objects must be of one class or another – NULL classes are not allowed.

The syntax of *cinemo_mm_classes* is as follows:

```
<cinemo_mm_classes>
    <class id="object">
        <class id="item">
            <!-- optional child classes -->
        </class>
        <class id="container">
            <!-- optional child classes -->
        </class>
    </class>
</cinemo_mm_classes>
```

Classes may be hierarchically arranged in a tree structure. Child classes are derived from parent classes. Class derivation can be tested in queries using the *derivedfrom* operator, which returns the boolean result *true* if a specified class is derived from another specified class. All classes are derived from the *object* class.

Cinemo MM, in accordance with UPnP specification, requires that certain built-in classes are correctly defined. These are the minimum set of classes required to run a UPnP media server – Cinemo MM will fail to start if these classes are missing from the configuration file. These are as follows:

- object
- object.item
- object.container
- object.container.storageVolume
- object.container.storageFolder

Note, *object.container.storageVolume* and *object.container.storageFolder* are used to represent volumes and the real or virtual folders within indexed or virtual volumes. Note also that *object.item.audioItem*, *object.item.videoItem* and *object.item.imageItem* classes, although usually defined in media servers, are not mandatory. Removing these classes from the configuration file will disable Cinemo MM support for these classes.

The class of an object is stored in its *upnp:class* field, accessed as [CINEMO_METANAME_VFS_UPNP_CLASS](#) with [ICinemoMetapool](#). An application may define additional classes, which can be directly assigned to items in virtual volumes, or automatically assigned to hierarchy nodes in the rules for hierarchy generation. All classes may be used in queries – for example, the expression “*upnp:class derivedfrom object*” will return all objects in the UPnP hierarchy, since all items and containers are derived from *object*, but “*upnp:class derivedfrom object.item.application_specific.class*” will only return items of this application specific class.

2.10.2 Fields

All fields recognized by Cinemo MM are defined in the *cinemo_mm_fields* section of the configuration XML. All defined fields are searchable, which means they can be used in query expressions. All fields are sortable, which means they can be used in sort criteria. All fields are accessible through ICinemoMetapool, and all fields are persistent – that is, stored in the database for indexed and persistent virtual volumes.

The syntax of *cinemo_mm_fields* is as follows:

```
<cinemo_mm_fields>
  <field
    id="dc:title"
    type="text"
    metaname="Title"
    sortedby="cinemo:titleSortOrder"
    fallback="cinemo:name"
    phonetic="custom:titlePhonetic"
    value="${expression}"
    update="onEdit"
    groupmask="0x1">
  </field>
</cinemo_mm_fields>
```

- ***id***

The UPnP field identifier, in this example, dc:title.

- ***type***

Must be one of text, ui32 or ui64.

- ***metaname***

The **CINEMO_METANAME** key representing this field in **ICinemoMetapool**. In this example, the key is 'Title', which is defined by **CINEMO_METANAME_TITLE**.

- ***sortedby***

Optional alternative field for sorting. In this example, *dc:title* is sorted by *cinemo:titleSortOrder*, which means that when the sort criteria for a browse or search operation is *dc:title*, for each object sorted, Cinemo will attempt to use *cinemo:titleSortOrder* instead, and only use *dc:title* if *cinemo:titleSortOrder* is not present.

- ***fallback***

Optional fallback field. In this example, *dc:title* has fallback *cinemo:name*, which means that for each object in browse or search results, if no *dc:title* metadata is present, the contents of *cinemo:name* will be returned as *dc:title* instead.

- ***phonetic***

Optional phonetic field. In this example, *custom:titlePhonetic* holds the phonetic representation for the title. The phonetic representation is used for phonetic matching.

- ***value***

Optional value. The value allows to define fields that are extracted from existing metadata based on expressions.

- **update**

Optional update method. Update method to be used for the value, either *onEdit*, *onAccess*, or *onReturn*.

- **groupmask**

Optional groupmask. Assigns groups to fields. The default group is 0x1.

Care should be taken when customizing *cinemo_mm_fields*, that all fields referred to in the optional *fallback* and *sortedby* attributes of a field actually exist. Cinemo MM will fail to start if the configuration file is invalid. It is possible to define fields without specifying the *metaname* attribute. In this case, the field will not be readable or writeable with [ICinemoMetapool](#), but can still be assigned in the rules for hierarchy nodes. Further, care should be taken when using the *value* and *update* attributes, since they can effect the indexing and query performance. Please contact your Cinemo representative if you want to use this feature.

The optional groupmask attribute allows to specify a group membership for the field. Overall 32 groups are supported. If CinemoMM is browsed or searched via [ICinemoPlaylist](#) using a Cinemo client, the [CINEMO_EC_PLAYLIST_CHANGED](#) event will forward the changed groups as its fourth parameter. This information may be used to reduce the amount of events that need to be handled by the client application.

2.10.3 Volume Groups

Cinemo MM will construct an initial hierarchy of volume groups based on the *cinemo_mm_root* section of the configuration XML. The UPnP root node, identified by *cinemo_mm_root*, is itself a volume group, whose initial fields and hierarchy can be defined, and which can contain optional child volume groups.

The syntax of *cinemo_mm_root* is as follows:

```
<cinemo_mm_root hierarchy="root">
    <container id="1">
        <field id="cinemo:name" value="Volumes"/>
        <field id="upnp:class" value="object.container"/>
        <field id="upnp:albumArtURI" value="serve://res://mm_volume.png?mimetype=image/png"/>
    </container>
</cinemo_mm_root>
```

The default Cinemo MM configuration, as declared above, will construct a single ‘Volumes’ container as a child of the root node, with a fixed object ID of ‘1’, and with a defined *cinemo:name*, *upnp:class* and *upnp:albumArtURI*. An application is free to change this behaviour.

In the above example, the *cinemo_mm_root* node is declared with a specific hierarchy identifier, which refers to the set of rules with the same ID, documented in the next section. All volume groups may be created with an optional hierarchy identifier, in which case the defined hierarchy is automatically created as children of the volume group. An alternative example is given below:

```
<cinemo_mm_root>
    <container id="1" hierarchy="hierarchy 1">
        <field id="cinemo:name" value="My Custom USB Devices"/>
    </container>
    <container id="2" hierarchy="hierarchy 2">
        <field id="cinemo:name" value="My Custom Mass Storage Devices"/>
    </container>
</cinemo_mm_root>
```

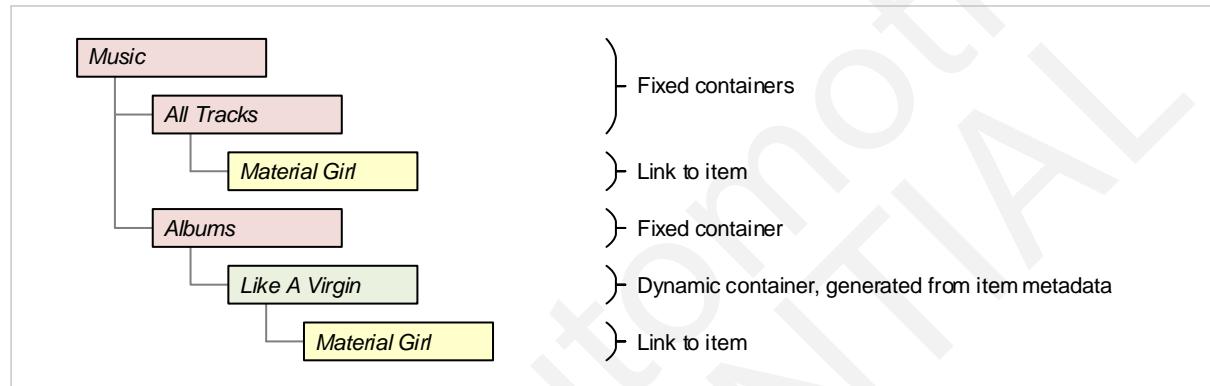
This example constructs a volume group hierarchy with two additional levels, and requires two custom hierarchies to be defined, with identifiers “*hierarchy 1*” and “*hierarchy 2*”. Volumes can be created within group “1” or group “2”, and the respective hierarchies will be generated only from content in one or the other group. As can be seen, the XML is flexible enough to support many alternative configurations.

2.10.4 Hierarchy Rules

Cinemo MM hierarchy nodes are automatically generated in accordance with a set of *hierarchy rules*, which are specified by identifier in *cinemo_mm_hierarchy* sections of the configuration XML. Hierarchy nodes consist of the following types of object:

- Fixed containers.
- Dynamic containers generated from item metadata.
- Links to items in volumes.

The following demonstrates a simple hierarchy with the syntax required to achieve it:



```

<cinemo_mm_hierarchy id="root">
    <container fixed="1" filter="$(upnp:class derivedfrom object.item.audioItem)">
        <field id="cinemo:name" value="Music"/>
        <container fixed="1">
            <field id="cinemo:name" value="All Tracks"/>
            <links/>
        </container>
        <container fixed="1" filter="$(upnp:album exists true)">
            <field id="cinemo:name" value="Albums"/>
            <container role="Album">
                <field id="cinemo:name" value="$(upnp:album)"/>
                <links/>
            </container>
        </container>
    </container>
</cinemo_mm_hierarchy>
  
```

The *id* attribute of the root *cinemo_mm_hierarchy* element is the identifier of the hierarchy, referred to by the *hierarchy* attribute of volume groups. Multiple hierarchies may be defined in the configuration XML, each with a unique identifier. The only way to create a node hierarchy is to create a volume group whose *hierarchy* attribute refers to it. Volume groups can be declared statically in the configuration XML, or created dynamically by means of [ICinemoMM::AddVolumeGroup\(\)](#). In both cases, the hierarchy must be specified when creating the volume group – it is not possible to dynamically attach a node hierarchy to an existing volume group.

The *fixed* attribute of a container specifies that the container will be created when the volume group is created, and will exist as long as the volume group exists. There are some restrictions to the rules for assigning field

values to fixed containers – in particular, the field assignment must be *static*, that is, it must not reference the metadata fields of any other item, since fixed containers are created before items are evaluated. If the *fixed* attribute is not specified, then the container is a *dynamic* container, which is constructed when items are evaluated, and automatically removed when it no longer contains any children.

The *filter* attribute of containers within the hierarchy is used to *exclude* items from the hierarchy. It must be defined as a UPnP query expression which operates on the fields of an item and that returns a boolean result – if the result is false then an item is excluded. In the above example, the entire hierarchy is constructed for “Music”, so it is desired to exclude all non-music items such as video and images from the hierarchy. In the case of the “Albums” sub-container, it is also desired to exclude music items for which no *Album* metadata exists.

The optional *role* attribute of containers within the hierarchy allows to assign an identifier to containers for retrieval via the [ICinemoMM::GetHierarchyNodes\(\)](#) API call.

Each container specifies a variable number of *field* assignments. Any field defined in the Cinemo MM configuration may be assigned to the generated container. Note, the *cinemo:name* field must always be assigned, since it is a requirement of Cinemo MM that all items and containers in the UPnP hierarchy are named. The above example has been simplified – in actual use, it is recommended to assign the *upnp:class* and *upnp:searchClass* fields to all containers in the node hierarchy.

There are two types of field assignment, *static* and *dynamic*. A static assignment assigns a static value to the field, for example, *cinemo:name* = “Albums”. A dynamic assignment, on the other hand, is indicated by the ‘\$’ symbol and assigns the *evaluated result of an expression* to the field, where the expression is evaluated on items. For example, the dynamic assignment *cinemo:name* = “\$*upnp:album*” will assign the value of an item’s *upnp:album* field to the container’s name. Expressions can be complex, constructing new field values from arbitrary combinations of item metadata fields. For example, *cinemo:name* = “\$*cinemo:videoCx* + “x” + *cinemo:videoCy*” will assign a name of the pattern “320 x 240”, based on the X and Y resolution of a video or image item. More complex examples can be found in the default Cinemo MM configuration file.

Finally, the *links* element specifies where links to items are created. A container which contains *links* cannot also contain child containers – that is, links are always leaf nodes in the hierarchy. Furthermore, it is a requirement that *all* leaf nodes are links. **It is not valid to construct a node hierarchy consisting of only containers** – all containers must ultimately lead to links.

2.10.5 Hierarchy Building

Cinemo MM hierarchy nodes are automatically generated in accordance with a set of *hierarchy rules* defined in the previous section. This section describes how the rules are actually applied.

Items can be added and removed to volumes, and their metadata fields can be edited at any time. This occurs automatically by Cinemo MM during the indexing of indexed volumes, or as a direct result of various methods which may be called on indexed and virtual volumes, for example [ICinemoMM::AddVirtualFile\(\)](#), [ICinemoMM::EditNode\(\)](#) and [ICinemoMM::RemoveNode\(\)](#). When an item is added to a volume, it will be *linked* – that is, the hierarchy rules for all parent volume groups of the volume will be evaluated on the item. When an item is removed from a volume, it will be *unlinked* – that is, all links to the item will be removed from the

hierarchy nodes in which they were created. When an item is edited, it is first unlinked, and then linked again – since the metadata fields of an item may change when editing the item, the hierarchy rules need to be re-evaluated.

The process of linking an item is straight-forward:

1. The volume group hierarchy is traversed in an upward direction, starting from the immediate parent of the item's volume, which must be a volume group, and ending at the root node, which is also a volume group. For each volume group with a specified hierarchy attribute, the corresponding hierarchy rules are evaluated. Volume groups with no hierarchy will be simply be skipped.
2. Hierarchy rules are traversed from top to bottom in depth-first order. If the *filter* attribute of a hierarchy container is declared, then the filter expression is first evaluated on the item, and the container will only be traversed if the boolean result of the filter expression is *true*.
3. The field assignments for each hierarchy container are evaluated based on the metadata fields of the item. If no container already exists at the current location with the same field values, a new container is created. Note that a container with a particular combination of field values **can be created only once** at its defined location in the node hierarchy. This prevents, for example, multiple containers with the same *Album* name appearing under the *Albums* container. It does not prevent, however, multiple containers with the same *Album* name but different values for another field, e.g. *AlbumArtist* – a new container will always be created unless all assigned fields are **identical** to an existing container at the same location. The only exception, where an existing container is reused even if assigned fields differ, is made for fields having the *merge* attribute set. In this case, the metadata of items will be merged according to the merge strategy *simple* or *unique* into the field of an existing hierarchy container. The simple merge strategy will set the field of the hierarchy container based on the metadata of the last item that passed the filter of the container. The unique merge strategy generates a unique list from the metadata of all filtered items with consecutive indexes in the field of the hierarchy container. Note that the number of entries in the unique list is limited to a maximum of 256 entries.
4. If the container is a leaf node – that is, contains a *links* element – then a link will be created to the item at the current location. The linking process is complete.

The process of *unlinking* an item is the reverse of the linking process. All links to the item will be removed from the node hierarchy, and any resulting empty containers with no *fixed* attribute will also be removed. If all items in a volume group are unlinked, the resulting hierarchy will consist only of empty *fixed* containers, equal to the state when the volume group was first created.

2.11 Locales

Cinemo MM makes use of language sensitive operations for searching, collation, and sorting:

- Searching
searching is used for substring matching operators such as *contains* and *startswith*
- Collation
collation is used for the operator *equals*
- Sorting
sorting requires language dependent collation

Cinemo MM allows adapting these language specific operations by configuring an appropriate locale. Depending on the application requirements, Cinemo MM can be configured to use a single locale or to offer multiple locales for supporting different languages at the same time.

Please note that a multi language configuration requires additional memory and processing resources. This is usually only justifiable if multiple languages need to be supported at the same time. Please discuss with your Cinemo representative before using this feature in your product.

2.11.1 Single Locale Configuration

If only a single locale shall be supported, the language support can be configured using the option `CINEMO_OPTION_MM_ICU_MODE`.

If the ICU mode is set to *Disable*, Cinemo MM will use a fast, language independent [Unified Collation Algorithm \(UCA\)](#) implementation and a language independent [Case Folding](#) for collation and sorting.

If the ICU mode is set to *Sort*, Cinemo MM will use the ICU library for collation and sorting. The behavior can be controlled by the `CINEMO_OPTION_MM_ICU_SORT_*` options.

If the ICU mode is set to *SortSearch*, the ICU library will be used for searching as well. The search behavior can be configured using the `CINEMO_OPTION_MM_ICU_SEARCH_*` options.

From ICU version 55.1 on the locale string (`CINEMO_OPTION_MM_SORT_LOCALE` and `CINEMO_OPTION_MM_SEARCH_LOCALE`) can contain additional ICU options according to [Unicode Locale Data Markup Language \(LDML\)](#). For older versions of ICU, the options 'kn', 'ka', and 'kk' are supported by Cinemo MM and mapped to the corresponding ICU attributes. Please also note, that setting conflicting options in the ICU locale and the options `CINEMO_OPTION_MM_ICU_SORT_IGNORE`, `CINEMO_OPTION_MM_ICU_SORT_NUMERIC`, or `CINEMO_OPTION_MM_ICU_SEARCH_IGNORE` will result in a failure to initialize Media Management.

2.11.2 Multi Locale Configuration

Cinemo MM can be configure to support multiple locales using the locale configuration XML file. The path to the locale configuration file is passed to Cinemo MM via the option `CINEMO_OPTION_MM_LOCALES_CONFIG_XML`.

A locale in the configuration file has the following syntax

```
<cinemo_mm_locales version="1.0">
  <locale id="en">
    <collate>en_US-u-true-ka-shifted</collate>
    <search></search>
  </locale>
</cinemo_mm_locales>
```

Each Cinemo MM locale is identified by its *id*. The *collate* value is used to configure the locale for sorting and comparison. The *search* value is used to configure the locale for substring searching. The syntax of the values follows the [Unicode Locale Data Markup Language \(LDMF\)](#) format. An empty *collate* configures Cinemo MM to use a fast, language independent [Unified Collation Algorithm \(UCA\)](#) implementation. An empty *search* value will use language independent [Case Folding](#).

In the above example the *collate* locale is configured as *en_US-u-true-ka-shifted* using the 'en' locale with 'US' collation and the options numeric sorting (kn-true) and shift ignorables (ka-shifted). Note that the complete set of available options is supported from ICU version 55.1 and higher. For older versions, only the options 'kn', 'ka', and 'kk' are supported by Cinemo MM.

The locale configuration file allows to define up to 255 distinct locales.

By default, Cinemo MM extracts a locale dependent sortkey for efficient sorting and collation. A sortkey will be extracted for locales defined in the option *CINEMO_OPTION_MM_LOCALES_AVAILABLE*. These locales will be available for efficient client queries.

The option *CINEMO_OPTION_MM_LOCALES_COLLATION_FALLBACK* configures Cinemo MM to fallback to collation without sortkey if the queried locale is not in the list of available locales. Collation without sortkey is only recommended, if the platform has enough processing resources.

The option *CINEMO_OPTION_MM_LOCALES_DEFAULT* configures a default locale that is used if a Cinemo MM client does not specify a locale or if a 3rd party UPnP client is used.

Locale Selection

By default, Cinemo MM clients will use the locale defined under the *CINEMO_OPTION_MM_LOCALES_DEFAULT* option on the Cinemo MM server. In order to request a specific locale, the additional URL parameter *locale* needs to be provided:

```
SampleMetadata "upnp://192.168.0.1:40000/0/MediaServer/DeviceDesc.xml?pid=<media_pid>&browse=BrowseDirectChildren&locale=<locale_id>"
```

The list of locales available on the server can be retrieved from the device description under the metaname *CINEMO_METANAME_VFS_UPNP_LOCALES*:

```
SampleMetadata "upnp://192.168.0.1:40000/0/MediaServer/DeviceDesc.xml"
```

3. Cinemo Interfaces

All Cinemo interfaces are defined in C++ header files which are provided with the Cinemo Media Engine SDK. These headers are pre-processed based on the enabled feature set for a project, and so may not include all the interfaces and API functions documented here. For example, [ICinemoMM](#) interface is only available when Cinemo Media Management feature is enabled. If you require an interface or function which is described in this document but is missing from the Cinemo SDK headers, then please contact your Cinemo representative to provide you an updated SDK with these features enabled.

All Cinemo interfaces are defined in C++ and follow the Component Object Model (COM) specification. As defined by COM, all Cinemo interfaces derive from a standard [ICinemoUnknown](#) interface, which consists of three methods:

- [ICinemoUnknown::AddRef\(\)](#) and [ICinemoUnknown::Release\(\)](#), reference counting to control the lifetime of interfaces.
- [ICinemoUnknown::QueryInterface\(\)](#), access to interfaces by ID.

All Cinemo interfaces are reflexive, symmetric, and transitive. The reflexive property means that calling [ICinemoUnknown::QueryInterface\(\)](#) on a given interface with the interface's ID will return the same instance of the interface. The symmetric property means that when interface B is retrieved from interface A by calling [ICinemoUnknown::QueryInterface\(\)](#), interface A is also retrievable from interface B. The transitive property means that if interface B is obtainable from interface A and interface C is obtainable from interface B, then interface C is also retrievable from interface A.

3.1 ICinemoUnknown

All Cinemo interfaces are derived from [ICinemoUnknown](#). This interface is thread-safe.

ICinemoUnknown Methods

METHOD	DESCRIPTION
AddRef	Increment reference count
Release	Decrement reference count, delete if the count becomes zero
QueryInterface	Query this object for an interface

3.1.1 AddRef

Increment the reference count on an interface.

Syntax

```
int AddRef();
```

Return value

An integer value indicating the current reference count.

Remarks

The returned reference count is guaranteed to be greater than zero.

3.1.2 Release

Decrement the reference count on an interface, and delete the interface when the reference count becomes zero.

Syntax

```
int Release();
```

Return value

An integer value indicating the current reference count.

Remarks

If the returned reference count is zero, the interface was automatically deleted and its functions should no longer be called. When an interface is deleted, its underlying object and associated resources are freed.

3.1.3 QueryInterface

Query for an interface with the specified interface ID.

Syntax

```
CinemoError QueryInterface(  
    [out] void ** pp,  
    [in] const char * iid  
) ;
```

Parameters

- **pp**
The address of a pointer that receives the requested interface.
- **iid**
The identifier of the requested interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, *pp[0]* will point to the returned interface. The reference count of the returned interface is automatically incremented. Calling applications should [Release\(\)](#) the interface when it is no longer required.

All Cinemo interfaces have a unique interface ID, which can be coded simply by appending “::iid” to the name of the interface. For example:

```
ICinemoUnknown * p = NULL;  
QueryInterface((void **) &p, ICinemoUnknown::iid);
```

In the above example, it is not necessary to check the return value of [QueryInterface\(\)](#), because all Cinemo objects expose the [ICinemoUnknown](#) interface. As specified by COM, it is possible to use this property to check for the identity of interfaces, since querying different interfaces of the same object for [ICinemoUnknown](#) must return identical pointers.

3.2 ICinemoUTF8

The [ICinemoUTF8](#) interface is used to manage text strings. The methods of many Cinemo interfaces return UTF-8 text strings. The text content of any object exposing the [ICinemoUTF8](#) interface is guaranteed to be a valid UTF-8 character string. The interface is intended to simplify tasks concerning the memory management of strings. Please note that this interface is not thread-safe.

Notes

Many Cinemo API methods require the application to provide a pre-allocated [ICinemoUTF8](#) interface for returning strings, other methods will allocate their own [ICinemoUTF8](#), and require the application to release it when finished with the string. These decisions are made for performance reasons: if a function (which returns a string) is not expected to be called often, then it is more convenient for the calling application if the function allocates the string. On the other hand, if a function is expected to be called frequently, then it is better for performance reasons if the application pre-allocates a [ICinemoUTF](#) object, and re-uses its interface every time the function is called.

In cases where the calling application is required to pre-allocate a string, it can do this with the [CinemoCreateUTF8\(\)](#) function. To prevent memory leaks, an application should always call [ICinemoUnknown::Release\(\)](#) on a string interface which is no longer required.

ICinemoUTF8 Methods

METHOD	DESCRIPTION
Assign	Assign a UTF-8 string
Format	Format a UTF-8 string using <code>printf()</code> syntax
FormatMediaType	Format a UTF-8 string describing the given media type
Clear	Clear string contents
Ptr	Get a pointer to the current string
Ptrz	Get a non-NULL pointer to the current string
Bytes	Get the string length in bytes
Chars	Get the string length in characters
Cmp	Case sensitive comparison
Cmpi	Case insensitive comparison
Cmpn	Case sensitive comparison (limit bytes)
Cmpni	Case insensitive comparison (limit bytes)
Atoi	Convert string to 32-bit integer
Atoi64	Convert string to 64-bit integer
SetURLParameter	Set a URL parameter.
GetURLParameter	Get a URL parameter.

3.2.1 Assign

Assign a UTF-8 string.

Syntax

```
const char * Assign(
    [in] const char * szchars,
    [in] int nbytes
);
```

Parameters

- **szchars**

Pointer to a UTF-8 character string.

- **nbytes**

The length of the string in bytes.

Return value

A pointer to the string, if valid, otherwise NULL.

Remarks

The string will be parsed to ensure it is valid UTF-8. Only those characters which are valid UTF-8 will end up in the assigned string (other characters will be ignored). If the length of the string is unknown, it may be specified as -1. In this case, the string must be zero terminated. The returned pointer is valid until either the [ICinemoUTF8](#) interface is released, or until another method is called which changes the string contents.

3.2.2 Format

Format a UTF-8 string using [printf\(\)](#) style syntax.

Syntax

```
const char * Format(  
    [in] const char * szchars,  
    [in] ...  
) ;
```

Parameters

- **szchars**

Pointer to a UTF-8 character string.

Return value

A pointer to the string, if valid, otherwise NULL.

Remarks

The string will be parsed to ensure it is valid UTF-8. Only those characters which are valid UTF-8 will end up in the assigned string (other characters will be ignored). The returned pointer is valid until either the [ICinemoUTF8](#) interface is released, or until another method is called which changes the string contents.

3.2.3 FormatMediaType

Format a UTF-8 string to describe a CinemoMediaType.

Syntax

```
const char * FormatMediaType (
    [in] const CinemoMediaType * p
);
```

Parameters

- **p**

Pointer to a CinemoMediaType structure.

Return value

A pointer to the string, if valid, otherwise NULL.

Remarks

This method is intended for convenience. The returned string will fully describe the media type. The returned pointer is valid until either the [ICinemoUTF8](#) interface is released, or until another method is called which changes the string contents.

3.2.4 Clear

Clear string contents.

Syntax

```
void Clear();
```

Remarks

Reset the string and free its memory.

3.2.5 Ptr

3.2.6 Ptrz

Get a pointer to the string.

Syntax

```
const char * Ptr();  
const char * Ptrz();
```

Return value

Return a read-only pointer to the UTF-8 string. In the case of [Ptr\(\)](#), the return value is NULL if the string is empty. In the case of [Ptrz\(\)](#), the return value is never NULL. If the string is empty, [Ptrz\(\)](#) will return a pointer to a zero-terminated empty string ("\"0").

Remarks

The returned pointer is valid until either the [ICinemoUTF8](#) interface is released, or until another method is called which changes the string contents.

3.2.7 Bytes

Get the length of the string in bytes.

Syntax

```
uint32 Bytes();
```

Return value

An integer value ≥ 0 , indicating the length of the string in bytes.

Remarks

Returns zero if the string is empty.

3.2.8 **Chars**

Get the length of the string in UTF-8 characters.

Syntax

```
int Chars();
```

Return value

An integer value ≥ 0 , indicating the length of the string in UTF-8 characters.

Remarks

Returns zero if the string is empty. The length of the string in UTF-8 characters will always be less than or equal to its length in bytes, since a UTF-8 character may occupy 1 to 4 bytes.

3.2.9 Cmp

3.2.10 Cmpi

3.2.11 Cmpn

3.2.12 Cmpni

Compare with another UTF-8 string.

Syntax

```
int Cmp(
    [in] const char * szchars
);
int Cmpi(
    [in] const char * szchars
);
int Cmpn(
    [in] const char * szchars,
    [in] int nbytes
);
int Cmpni(
    [in] const char * szchars,
    [in] int nbytes
);
```

Parameters

- **szchars**

Pointer to a UTF-8 character string.

- **nbytes**

Length of the string, in bytes.

Return value

Standard return values of the comparison function, which may be negative, positive, or zero if the strings are identical.

Remarks

The string to compare with must be zero terminated valid UTF-8.

3.2.13 Atoi**3.2.14 Atoi64**

Convert string to 32-bit or 64-bit integer value.

Syntax

```
sint32 Atoi();  
sint64 Atoi64();
```

Return value

Return the integer interpretation of the UTF-8 string.

Remarks

The methods are identical to standard C library functions [atoi\(\)](#) and [atol\(\)](#). The method skips as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, it parses an optional plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value. The string can contain additional characters after those that form the integral number, which are ignored.

If the string is empty, contains only whitespace characters, or if the first sequence of non-whitespace characters in the string is not a valid integral number, the method will return 0.

3.2.15 SetURLParameter

Add, change, or remove a URL parameter in the string.

Syntax

```
CinemoError SetURLParameter(  
    [in] const char * szname,  
    [in] const char * szvalue);
```

Parameters

- **szname**

The URL parameter name.

- **szvalue**

The URL parameter value, or NULL.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will add a new URL parameter to the string, change an existing URL parameter, or remove an existing URL parameter.

The *szname* argument is the parameter name. This must consist entirely of RFC 3986 section 2.3 unreserved characters, that is, the characters A-Z, a-z, 0-9, - (hyphen), _ (underscore), . (dot) and ~(tilde). If any other characters are present, the method will return *CinemoErrorArguments*.

The *szvalue* argument specifies the parameter value. Any existing parameter values with the same name will be removed from the string. If set to non NULL, the new parameter will be inserted in the string. The parameter value will be *URL escaped* before insertion. This is to ensure conformance of the resulting URL string RFC 3986.

3.2.16 GetURLParameter

Get a URL parameter from the string.

Syntax

```
CinemoError GetURLParameter(  
    [in] const char * szname,  
    [out] struct ICinemoUTF8 ** pp);
```

Parameters

- **szname**

The URL parameter name.

- **pp**

The returned URL parameter value.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will retrieve a URL parameter from the string.

The *szname* argument is the parameter name. This must consist entirely of RFC 3986 section 2.3 unreserved characters, that is, the characters A-Z, a-z, 0-9, - (hyphen), _ (underscore), . (dot) and ~(tilde). If any other characters are present, the method will return *CinemoErrorArguments*.

If a URL parameter with the given name is found in the string, it will be *URL unescaped* and returned as a valid [ICinemoUTF8](#) string. Otherwise, the method will return *CinemoErrorNoSuchObject*.

3.3 ICinemoAudioCorrelator

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

The ICinemoAudioCorrelator interface provides access to the audio correlator for livestream correlation. It may be used for synchronizing playback of two livestream sources whose contents are identical—apart from an unknown time offset and potential differences in the stream formats.

ICinemoAudioCorrelator Methods

METHOD	DESCRIPTION
UpdateLiveStreamParams	Add correlator reference to ICinemoLiveStream parameters.

Configuration Parameters

Optional configuration parameters can be specified when creating an ICinemoAudioCorrelator instance. Parameters are passed as the *szparam* argument to [CinemoCreateAudioCorrelator\(\)](#) in format of URL parameters.

The following parameters are supported:

Parameter	Description	Example
range	Defines the time range in milliseconds over which audio correlation (i.e. similarity search) is performed for finding the effective playback time offset between the players. E.g. by specifying a range of 1000 ms, time offsets between the players from -500 ms to +500 ms can be detected. Default is 1000.	?range=1000
window	Defines the duration of the search window in milliseconds that is used as a sliding window over the range to calculate individual audio correlation results. The value should be chosen as a tradeoff between CPU performance and detection accuracy. Default is 50.	?window=50

Notes

For synchronizing playback of two livestreams, both [ICinemoLiveStream](#) instances need to be registered with an [ICinemoAudioCorrelator](#) instance on their creation. The sample code `SampleLiveStreamSwitch` shows how do so. When two livestream objects are registered with an audio correlator and data is received on both livestream objects, the audio correlator will implicitly try to find a correlation.

If a correlation is found, the playback position of the livestream that has later been registered with the audio correlator will automatically be adjusted such that both streams are synchronously played back. The [ICinemoPlayer](#) instance whose playback position was adjusted will receive a [CINEMO_EC_CORRELATION](#) event.

Switching between the same livestream from different sources likely will imply switching between players with different levels of buffering. Thus, the option [CINEMO_OPTION_LIVESTREAM_PREBUFFER_MS](#) should be chosen to reflect the maximum expected (or allowed) time difference between the earliest arriving signal and the latest arriving signal (as the “range” parameter should be as well). E.g. if playback is started after buffering 1000 ms and the target stream for switching is late by also 1000 ms, there will be no buffering room for switching left. [CINEMO_OPTION_LIVESTREAM_PREBUFFER_MS](#) should therefore be chosen to have enough buffering room left for all allowed time deviations of the input stream.

If the timing of the livestream input data prevents a correlation, this is indicated by a [CINEMO_EC_CORRELATION](#) event with error code *CinemoErrorUnderflow*.

3.3.1 UpdateLiveStreamParams

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Add correlator reference to ICinemoLiveStream parameters.

Syntax

```
CinemoError UpdateLiveStreamParams (
    [in/out] struct ICinemoUTF8 * pparams
);
```

Parameters

- **pparams**
ICinemoLiveStream parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.4 ICinemoBlob

The [ICinemoBlob](#) interface is used to manage blocks of binary data. The methods of many Cinemo interfaces return blobs (typically images, but any binary data is supported). The interface is intended to simplify tasks concerning the memory management of blobs. Please note that this interface is not thread-safe.

Notes

Some Cinemo API methods require the application to provide a pre-allocated [ICinemoBlob](#) interface for returning blobs, other methods will allocate their own [ICinemoBlob](#), and require the application to release it when finished with the blob. These decisions are made for performance reasons: if a function (which returns a blob) is not expected to be called often, then it is more convenient for the calling application if the function allocates the blob. On the other hand, if a function is expected to be called frequently, then it is better for performance reasons if the application pre-allocates a [ICinemoBlob](#) object, and re-uses its interface every time the function is called.

In cases where the calling application is required to pre-allocate a blob, it can do this with the [CinemoCreateBlob\(\)](#) function. To prevent memory leaks, an application should always call [Release\(\)](#) on a blob interface which is no longer required.

ICinemoBlob Methods

METHOD	DESCRIPTION
Assign	Assign a blob of data
AssignStatic	Assign a static constant blob of data
Load	Load blob from a file
Clear	Clear blob contents
Resize	Resize blob and get a writable pointer
Ptr	Return a pointer to the blob
Bytes	Return the blob size
Crc32	Return the blob CRC-32 value

3.4.1 Assign

Assign a blob of data.

Syntax

```
CinemoError Assign(
    [in]  const void * pdata,
    [in]  uint32 nbytes
);
```

Parameters

- **pdata**

Pointer to a binary block of data.

- **nbytes**

The length of the data in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Memory for the blob will be allocated internally, and the data copied. The application may free the original data since it is no longer required.

3.4.2 AssignStatic

Assign a static constant blob of data.

Syntax

```
CinemoError AssignStatic(  
    [in]  const void * pdata,  
    [in]  uint32 nbytes  
) ;
```

Parameters

- **pdata**

Pointer to a binary block of data.

- **nbytes**

The length of the data in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code. The method always succeeds.

Remarks

For performance reasons the memory for the blob will NOT be copied, the interface will instead maintain a pointer to the original blob. The application should take care not to free the data until it is no longer referenced by the interface.

3.4.3 Load

Load blob from a file.

Syntax

```
CinemoError Load(  
    [in]  const char * szfilename  
);
```

Parameters

- **szfilename**

The name of the file to load

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The method will use standard operating system functions, e.g. [open\(\)](#) and [read\(\)](#), to load the file. The entire file is loaded into memory, so it is advised to check the file size before calling this method.

3.4.4 Clear

Clear blob contents.

Syntax

```
void Clear();
```

Remarks

Reset the blob, and free any allocated memory.

3.4.5 Resize

Resize the blob and get a writable pointer.

Syntax

```
void * Resize(int nbytes);
```

Parameters

- **nbytes**

The length of the data in bytes.

Return value

If the method succeeds, it returns a non-NULL pointer to a block of uninitialized memory which may be written to by the application. Otherwise, it returns NULL.

Remarks

The implementation internally uses [realloc\(\)](#).

3.4.6 Ptr

Get a pointer to the blob.

Syntax

```
const void * Ptr();
```

Return value

A pointer to the blob, if valid, otherwise NULL.

Remarks

The returned pointer is valid until either the [ICinemoBlob](#) interface is released, or until another method is called which changes the blob contents.

3.4.7 Bytes

Get the length of the blob in bytes.

Syntax

```
uint32 Bytes();
```

Return value

An integer value ≥ 0 , indicating the length of the blob in bytes.

Remarks

Returns zero if the blob is empty.

3.4.8 Crc32

Return the blob CRC-32 value.

Syntax

```
uint32 Crc32();
```

Return value

The CRC-32 of the blob.

Remarks

The return value is zero if the blob is empty. The CRC-32 may be used to compare two blobs.

3.5 ICinemoMetaRawText

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

The [ICinemoMetaRawText](#) interface is used to manage raw text metadata that has not been processed by Cinemo. Please note that this interface is not thread-safe.

Notes

The calling application is required to pre-allocate the [ICinemoMetaRawText](#) interface. It can do this with the [CinemoCreateMetaRawText\(\)](#) function. To prevent memory leaks, an application should always call [Release\(\)](#) on this interface when it is no longer required.

ICinemoMetaRawText Methods

METHOD	DESCRIPTION
Assign	Assign raw text metadata from an opaquely encoding string.
Clear	Clear contents.
Ptr	Return a pointer to the raw metadata text.
Bytes	Return the raw metadata text size.
Srctype	Return type of metadata source.
Hint	Return a character set hint.
Bpc	Return a bytes per character hint.

3.5.1 Assign

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Assign raw text metadata from an opaquely encoded string. The string can be retrieved from an [ICinemoMetapool](#) interface.

Syntax

```
CinemoError Assign(  
    [in] const char * szchars,  
    [in] uint32 nbytes  
) ;
```

Parameters

- **szchars**

Pointer to encoded raw text metadata.

- **nbytes**

The length of the data in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*.

If the input data does not contain raw text metadata, it returns *CinemoErrorNoSuchObject*. Then you should continue processing the metadata like usual processed metadata text.

Otherwise, it returns a *CinemoError* code.

3.5.2 Clear

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Clear contents.

Syntax

```
void Clear();
```

Remarks

Reset the raw text metadata, and free any allocated memory.

3.5.3 Ptr

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Get a pointer to the raw metadata text.

Syntax

```
const void * Ptr();
```

Return value

A pointer to the raw metadata text, if valid, otherwise NULL.

Remarks

The returned pointer is valid until either the [ICinemoMetaRawText](#) interface is released, or until another method is called which changes the raw metadata text contents.

3.5.4 Bytes

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Get the length of the raw metadata text in bytes.

Syntax

```
uint32 Bytes();
```

Return value

An integer value ≥ 0 , indicating the length of the raw metadata text in bytes.

Remarks

Returns zero if the raw metadata text is empty.

3.5.5 Srctype

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Return the type of source for this metadata raw text like ID3v1, ID3v2, etc.

Syntax

```
const char *Srctype();
```

Return value

One of the following values will be returned:

Value	Meaning
ID3v1	Metadata originates from ID3v1 tag.
ID3v2	Metadata originates from ID3v2 frame.
MP4	iTunes Atom
MP4_3GPP	3GPP Atom
MP4_RIFF	RIFF AVI Atom
cdtext	CD-Text

Remarks

The returned pointer is valid until either the [ICinemoMetaRawText](#) interface is released, or until another method is called which changes the raw metadata text contents.

3.5.6 Hint

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Return the raw metadata text character set hint.

Syntax

```
const char * Hint();
```

Return value

The name of the regular charset given by the raw metadata. One of the following values will be returned:

Value	Character Set
iso-8859-1	Latin 1 (ISO-8859-1)
utf-8	UTF-8 Unicode.
utf-16	UTF-16 Unicode including a byte order mark.
utf-16-be	UTF-16 Unicode big endian possibly not including a byte order mark.
MS-JIS	Code page 932 from CD-Text
unknown	?

Remarks

The returned pointer is valid until either the [ICinemoMetaRawText](#) interface is released, or until another method is called which changes the raw metadata text contents.

3.5.7 Bpc

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Return the raw metadata text bytes per character hint.

Syntax

```
int Bpc();
```

Return value

One of the following values will be returned:

Value	Meaning
0	Single byte encoding (e.g. ISO-8859-x)
1	UTF-8 multibyte encoding
2	Two byte encoding (e.g. UTF-16)
3	Four byte encoding (e.g. UTF-32)

3.6 ICinemoDataPort

The [ICinemoDataPort](#) interface provides low-level methods for data access. Please note that this interface is only thread-safe with regards to Cancel/Enable.

It can be created using the *CinemoCreateDataPort()* function defined in *cinemo.h*.

ICinemoDataPort Methods

METHOD	DESCRIPTION
<u>Open</u>	Open the data port
<u>Close</u>	Close the data port
<u>Read</u>	Read from the data port
<u>Write</u>	Write to the data port
<u>Cancel</u>	Cancel any Read or Write calls
<u>Enable</u>	Enable any Read or Write calls after a Cancel

3.6.1 Open

Open the data port.

Syntax

```
CinemoError Open(
    [in] const char * szurl,           /* dataport url */
    [in] const char * szparam,         /* additional parameters */
);
```

Parameters

- **szurl**
URL of the dataport.
- **szparam**
Additional parameters for the dataport.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is symmetric with [Close\(\)](#). An [ICinemoDataPort](#) instance may be opened and closed multiple times, even with different URLs. Calling [Open\(\)](#) on an already open [ICinemoDataPort](#) instance will first close the old instance and then open the new one.

An attempt to use an [ICinemoDataPort](#) instance which has not yet been opened will return *CinemoErrorFileOpen*.

3.6.2 Close

Close the data port.

Syntax

```
CinemoError Close();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling [Close\(\)](#) the allocated resources will be released and all operations on the dataport will return *CinemoErrorFileOpen*.

3.6.3 Read

Read data from the data port.

Syntax

```
CinemoError Read(
    [in] void * pbits,          /* read buffer */
    [in] uint32 nsize,         /* read size */
    [out] uint32 * pread      /* return bytes read */
);
```

Parameters

- **pbits**

Pointer to the read buffer.

- **nsize**

The size of the read buffer.

- **pread**

A pointer to receive the number of bytes actually read.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, at least some data has been read into the supplied read buffer. The caller should ensure the read buffer and read buffer sizes are set appropriately.

3.6.4 Write

Write data to the data port.

Syntax

```
CinemoError Write(  
    [in] const void * pbits,      /* write buffer */  
    [in] uint32 nsize,          /* write size */  
    [out] uint32 * pwritten     /* return bytes written */  
) ;
```

Parameters

- **pbits**

Pointer to the write buffer.

- **nsize**

The size of the write buffer.

- **pwritten**

A pointer to receive the number of bytes actually written.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, at least some data has been written from the supplied write buffer. The caller should ensure the write buffer and write buffer sizes are set appropriately.

3.6.5 Cancel

3.6.6 Enable

Cancel or enable all [Read\(\)](#) and [Write\(\)](#) methods.

Syntax

```
CinemoError Cancel();  
CinemoError Enable();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may be called to asynchronously cancel any thread blocked on [Read\(\)](#) or [Write\(\)](#) calls. In this case, [Read\(\)](#) or [Write\(\)](#) will return *CinemoErrorCancel*.

Note the cancel operation is sticky – after calling [Cancel\(\)](#), the [ICinemoDataPort](#) object will remain in a canceled state until [Enable\(\)](#) is called. This is to avoid potential race conditions. The correct sequence of calls to unblock a thread blocked in [ICinemoDataPort](#) is as follows:

1. Call [Cancel\(\)](#)
2. Wait for your thread which calls [Read\(\)](#) or [Write\(\)](#) to exit
3. Call [Enable\(\)](#)

3.7 ICinemoEncoder

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

The ICinemoEncoder class is used to maintain an encoder instance.

ICinemoEncoder Methods

METHOD	DESCRIPTION
<u>GetAudioDeviceName</u>	Update device_name in audio parameters to connect player output to this encoder
<u>GetVideoDeviceName</u>	Update device_name in video parameters to connect player output to this encoder

3.7.1 GetAudioDeviceName

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Update device_name in audio parameters to connect player output to this encoder.

Syntax

```
CinemoError GetAudioDeviceName (
    [in/out]          CinemoAudioParams& params
) ;
```

Parameters

- **params**
Audio device parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.7.2 GetVideoDeviceName

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Update device_name in video parameters to connect player output to this encoder.

Syntax

```
CinemoError GetVideoDeviceName (
    [in/out]      CinemoVideoParams& params
) ;
```

Parameters

- **params**

Video device parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.8 ICinemoFile

The [ICinemoFile](#) interface provides methods for file I/O. Please note that this interface is not thread-safe.

Cinemo SDK provides a built-in implementation of the [ICinemoFile](#) interface which can be instantiated by means of [CinemoCreateFile\(\)](#) function. In cases where the built-in functionality is not sufficient, it is possible for applications to provide their own implementation, by deriving a class from [ICinemoFile](#) interface and implementing all of its virtual methods.

ICinemoFile Methods

METHOD	DESCRIPTION
GetURL	Return a UTF-8 string containing the file's URL
GetContentType	Return a UTF-8 string describing the content type
GetCaps	Return the file capabilities
GetType	Return the file type
GetSize	Return the file size
GetDuration	Return the file duration
GetCacheHints	Return hints to optimize caching of the file
Open	Open the file
Close	Close the file
WakeDevice	Wake or spin-up the device on which the file is stored
Cancel	Asynchronous cancel read and write operations
Enable	Asynchronous enable read and write operations
Read	Read from the file
Write	Write to the file
SetSize	Set the file size
Flush	Flush buffers
Reconnect	Reconnect to the server

3.8.1 GetURL

Return a pointer to a UTF-8 string containing the file's URL.

Syntax

```
const char * GetURL();
```

Return value

A UTF8 string that contains the file's URL.

Remarks

The returned URL should be identical to the URL used to open the file.

3.8.2 GetContentType

Return a pointer to a UTF-8 string describing the content type.

Syntax

```
const char * GetContentType();
```

Return value

The function returns a zero-terminated UTF-8 string. If [Open\(\)](#) has not been called, the return value will be NULL.

Remarks

This method returns the MIME type of the file contents, if known, otherwise NULL.

3.8.3 GetCaps

Return the capabilities of the file.

Syntax

```
uint32 GetCaps();
```

Return value

The return value is a bitmask of *CINEMO_FILE_CAPS* enumerated type. If [Open\(\)](#) has not been called the return value will be zero.

Remarks

The *CINEMO_FILE_CAPS* specifies which operations on the file are permitted, as described below:

```
typedef enum {
    CINEMO_FILE_CAPS_SEEK      = 0x00000001,      /* seek in bytes */
    CINEMO_FILE_CAPS_SEEKZERO  = 0x00000002,      /* seek to zero is possible */
    CINEMO_FILE_CAPS_SIZE       = 0x00000004,      /* size is known */
    CINEMO_FILE_CAPS_DURATION  = 0x00000008,      /* stream duration is known */
    CINEMO_FILE_CAPS_CANCEL    = 0x00000010,      /* cancel/enable supported */
    CINEMO_FILE_CAPS_RECONNECT = 0x00000040,      /* reconnect is possible */
} CINEMO_FILE_CAPS;
```

- **CINEMO_FILE_CAPS_SEEK**

The file supports seeking.

- **CINEMO_FILE_CAPS_SEEKZERO**

The file supports seeking to the beginning of the file.

- **CINEMO_FILE_CAPS_SIZE**

The file size is known.

- **CINEMO_FILE_CAPS_DURATION**

The file duration is known.

- **CINEMO_FILE_CAPS_CANCEL**

The file supports asynchronous cancel of read/write calls.

- **CINEMO_FILE_CAPS_RECONNECT**

The file supports reconnect operations with alternative HTTP headers.

3.8.4 GetType

Return the type of the file.

Syntax

```
uint32 GetType();
```

Return value

The return value is a bitmask of *CINEMO_VFS_TYPE* enumerated type. If [Open\(\)](#) has not been called the return value will be zero.

Remarks

The return value should be *CINEMO_VFS_TYPE_FILE*. No other values are currently supported.

3.8.5 GetSize

Return the file size in bytes.

Syntax

```
uint64 GetSize();
```

Return value

Returns the file size, in bytes, as a 64-bit unsigned integer. If [Open\(\)](#) has not been called the return value will be zero.

Remarks

This method returns *CINEMO_FILE_SIZE_UNKNOWN* if the file size is unknown or infinite (e.g. an infinite stream). In this case, the [GetCaps\(\)](#) method should not return *CINEMO_FILE_CAPS_SIZE*.

3.8.6 GetDuration

Return the file duration.

Syntax

```
pts70mhz GetDuration();
```

Return value

The return value is the duration of the file, in 70.56MHz units. If [Open\(\)](#) has not been called the return value will be zero.

Remarks

This method returns zero if the file duration is zero, unknown, or infinite. In this case, the [GetCaps\(\)](#) method should not return *CINEMO_FILE_CAPS_DURATION*.

3.8.7 GetCacheHints

Return information about the file page size.

Syntax

```
CinemoError GetCacheHints(  
    [out] CinemoFileCacheHints& cache  
) ;
```

Parameters

- **Cache**

Return the CinemoFileCacheHints structure.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a CinemoError code. If [Open\(\)](#) has not been called the return value will be *CinemoErrorFileOpen*.

Remarks

This method returns a CinemoFileCacheHints structure, containing information which may be used to optimize file caching algorithms:

```
typedef struct {  
    uint32 page_size;           /* page size in bytes */  
} CinemoFileCacheHints;
```

The page_size field indicates the recommended cache page size, which is the optimal block size for reading from the file. If there is no recommended page size, then page_size is zero.

3.8.8 Open

Open the file.

Syntax

```
CinemoError Open();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is symmetric with [Close\(\)](#). An [ICinemoFile](#) instance may be opened and closed multiple times during its lifetime, for example, if it is added to an [ICinemoPlaylist](#) using [ICinemoPlaylist::AppendSource\(\)](#) method and is played more than once.

The [ICinemoFile](#) instance returned by [CinemoCreateFile\(\)](#) must be opened before any other file operations such as read or write. An attempt to read a file which has not been opened will return *CinemoErrorFileOpen*.

3.8.9 Close

Close the file.

Syntax

```
CinemoError Close();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is symmetric with [Open\(\)](#). An [ICinemoFile](#) instance may be opened and closed multiple times during its lifetime, for example, if it is added to an [ICinemoPlaylist](#) using [ICinemoPlaylist::AppendSource\(\)](#) method and is played more than once.

After calling [Close\(\)](#), any operations on the file will return *CinemoErrorFileOpen*.

3.8.10 WakeDevice

Wake or spin-up the device on which the file is stored.

```
CinemoError WakeDevice();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code. If [Open\(\)](#) has not been called the return value will be *CinemoErrorFileOpen*.

Remarks

This method is intended to wake or spin-up the underlying storage device on which the file is stored. It may be called some time in advance of the [Read\(\)](#) method, to ensure there is no additional delay when reading the file, due to the storage device being powered down.

3.8.11 Cancel

3.8.12 Enable

Asynchronous cancel and enable read/write operations.

Syntax

```
CinemoError Cancel();  
CinemoError Enable();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code. If [Open\(\)](#) has not been called the return value will be *CinemoErrorFileOpen*.

Remarks

These methods will only be called if [GetCaps\(\)](#) returns *CINEMO_FILE_CAPS_CANCEL*, indicating that the file supports asynchronous cancel of read/write operations.

The [Cancel\(\)](#) method may be called to unblock any thread currently blocked inside [Read\(\)](#) or [Write\(\)](#) methods. In this case, the blocking method will return immediately with *CinemoErrorCancel*.

The [Cancel\(\)](#) method is sticky – after calling [Cancel\(\)](#), the [ICinemoFile](#) object will remain in a cancelled state until [Enable\(\)](#) is called. This is to avoid potential race conditions. The correct sequence of calls to unblock a thread blocked inside e.g. [Read\(\)](#) is as follows:

1. Call [Cancel\(\)](#),
2. Wait for your thread which calls [Read\(\)](#) to unblock,
3. Call [Enable\(\)](#).

3.8.13 Read

Read data from the file.

Syntax

```
CinemoError Read(
    [in] void * pbits,          /* read buffer */
    [in] uint64 nseek,         /* seek position */
    [in] uint32 nsize,         /* read size */
    [out] uint32 * pread      /* return bytes read */
);
```

Parameters

- **pbits**

Pointer to the read buffer.

- **nseek**

The seek position as an offset in bytes from the beginning of the file.

- **nsize**

The size of the read buffer.

- **pread**

A pointer to receive the number of bytes actually read, or NULL.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code. If [Open\(\)](#) has not been called the return value will be *CinemoErrorFileOpen*.

Remarks

If the method succeeds, at least some data has been read into the supplied read buffer. The caller should ensure the read buffer and read buffer sizes are set appropriately.

Note the behavior of this method is different depending if the pread argument is NULL or not NULL. If pread is NULL, then on successful return the number of bytes read will be equal to the size of the read buffer. If pread is not NULL, then on successful return it will contain the number of bytes actually read from the file, which will always be greater than zero, and less than or equal to the size of the read buffer.

3.8.14 Write

Write data to the file.

Syntax

```
CinemoError Write(
    [in] const void * pbits,      /* write buffer */
    [in] uint64 nseek,          /* seek position */
    [in] uint32 nsize,          /* write size */
    [out] uint32 * pwritten     /* return bytes written */
);
```

Parameters

- **pbits**

Pointer to the write buffer.

- **nseek**

The seek position as an offset in bytes from the beginning of the file.

- **nsize**

The size of the write buffer.

- **Pwritten**

A pointer to receive the number of bytes actually written, or NULL.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code. If [Open\(\)](#) has not been called the return value will be *CinemoErrorFileOpen*.

Remarks

If the method succeeds, at least some data has been written from the supplied write buffer. The caller should ensure the write buffer and write buffer sizes are set appropriately.

Note the behavior of this method is different depending if the pwritten argument is NULL or not NULL. If pwritten is NULL, then on successful return the number of bytes written will be equal to the size of the write buffer. If pwritten is not NULL, then on successful return it will contain the number of bytes actually written to the file, which will always be greater than zero, and less than or equal to the size of the write buffer.

The file must have been opened with *CINEMO_FILE_ACCESS_WRITE* permission.

3.8.15 SetSize

Set the file size.

Syntax

```
CinemoError SetSize(  
    [in]  uint64 nsize  
) ;
```

Parameters

- **nsize**

The new size of the file, in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code. If [Open\(\)](#) has not been called the return value will be *CinemoErrorFileOpen*.

Remarks

The [SetSize\(\)](#) method may be used to pre-allocate space on the underlying storage device for a file, or to truncate an existing file.

The file must have been opened with *CINEMO_FILE_ACCESS_WRITE* permission.

3.8.16 Flush

Flush read and write buffers.

Syntax

```
CinemoError Flush();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code. If [Open\(\)](#) has not been called the return value will be *CinemoErrorFileOpen*.

3.8.17 Reconnect

Reconnect to the file server.

Syntax

```
CinemoError Reconnect(  
    [in] const char * szheader  
) ;
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code. If [Open\(\)](#) has not been called the return value will be *CinemoErrorFileOpen*.

Remarks

This function reconnects to the file server in case the connection was broken. This is only possible for files with the *CINEMO_FILE_CAPS_RECONNECT* capability.

3.9 ICinemoLogConfig

The [ICinemoLogConfig](#) interface can be used to:

- obtain and modify the zone ID of an [ICinemoPlayer](#) or [ICinemoConfig](#) instance,
- generate custom log messages which will be output using the current zone ID and the currently configured log output target.

The zone ID can help to distinguish log output of the threads of one Cinemo player instance from the log output of other Cinemo player instances running in the same process. By default each new [ICinemoPlayer](#) instance is assigned a unique zone ID which is part of the Cinemo log messages of each player, e.g. "Z3" for zone ID 3. Please note that this interface is not thread-safe.

The [ICinemoLogConfig](#) interface can be obtained using the [ICinemoUnknown::QueryInterface\(\)](#) function on the [ICinemoPlayer](#) object.

ICinemoLogConfig Methods

METHOD	DESCRIPTION
GetZoneID	Get the zone ID of a Cinemo player instance.
SetZoneID	Set the zone ID of a Cinemo player instance.
GetMDC	Get the Mapped Diagnostic Context (MDC) of a Cinemo player instance.
SetMDC	Set a MDC key value pair of a Cinemo player instance.
LogMessage	Generate custom log message.

Notes

[SetZoneID\(\)](#) has to be called immediately after the creation of the object for which the zone ID should be changed. Until [SetZoneID\(\)](#) has been called, threads created by this object will inherit the previous zone ID which was assigned dynamically during object creation. The zone ID of these threads cannot be modified after creation.

3.9.1 GetZoneID

Get the zone ID of a Cinemo player instance.

Syntax

```
CinemoError GetZoneID(  
    [out] int& zone_id,  
);
```

Parameters

- **zone_id**

Reference to the variable that will be set to the current zone_id.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.9.2 SetZoneID

Set the zone ID of a Cinemo player instance.

Syntax

```
CinemoError SetZoneID(  
    [in] int zone_id,  
);
```

Parameters

- **zone_id**

The new zone_id.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.9.3 GetMDC

Get the Mapped Diagnostic Context (MDC) of a Cinemo player instance.

Syntax

```
CinemoError GetMDC(  
    [in] const char * szkey,  
    [out] struct ICinemoUTF8 ** ppval,  
) ;
```

Parameters

- **szkey**

The key string.

- **ppval**

The address of a ICinemoUTF8 pointer that will be filled with a ICinemoUTF8 instance that contains the value.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.9.4 SetMDC

Set a MDC key/value pair of a Cinemo player instance.

Syntax

```
CinemoError SetMDC(  
    [in] const char * szkey,  
    [in] const char * szval,  
) ;
```

Parameters

- **szkey**

The key string.

- **szval**

The value string associated to the respective key.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.9.5 LogMessage

Generate custom log message to be output by Cinemo.

Syntax

```
CinemoError LogMessage(  
    [in] CINEMO_LOGLEVEL level,  
    [in] const char * szid,  
    [in] const char * szformat,  
    ...  
) ;
```

Parameters

- **level**

Log level of the message.

- **szid**

Pointer to a UTF-8 character string used for human readable identification, e.g. the name of the module issuing this log message.

- **szformat**

Pointer to a UTF-8 character string with the log message.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.10 ICinemoLogAppender

The [ICinemoLogAppender](#) interface provides means for receiving and outputting log messages. It can be created using the *CinemoCreateLogAppender()* function defined in *cinemo.h*. It is allowed to have multiple ICinemoLogAppender instances active at the same time.

ICinemoLogAppender Methods

METHOD	DESCRIPTION
<u>Open</u>	Open a target for log output
<u>Open</u>	Open a callback target for getting log output
<u>Close</u>	Close a target and reset all settings
<u>SetLevel</u>	Set the verbosity level for all or a subset of log messages
<u>SetLayout</u>	Set the layout of log messages

3.10.1 Open

Open a target for log output. If a target is already open, the current one will be closed and the new one is opened using the current settings.

Syntax

```
CinemoError Open(
    [in] CINEMO_LOGTARGET target,
    [in] const char * szfilename = 0,
);
```

Parameters

- **target**

The target for writing log message.

- **szfilename**

Optional argument for setting the path to the log file and additional parameters that are written like URL parameters, i.e. added after a "?" and separated with the "&" character. Some valid parameters are listed below.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The Cinemo Media Engine generates log messages. This method gives you the ability to receive these log messages and output it to a specified destination.

The *CINEMO_LOGTARGET* enumeration is defined here:

```
typedef enum {
    CINEMO_LOGTARGET_NONE = 0,
    CINEMO_LOGTARGET_FILE,
    CINEMO_LOGTARGET_FILEDESCRIPTOR,
    CINEMO_LOGTARGET_PIPE,
    CINEMO_LOGTARGET_STDOUT,
    CINEMO_LOGTARGET_STDERR,
    CINEMO_LOGTARGET_CALLBACK,
    CINEMO_LOGTARGET_OUTPUT_DEBUG_STRING,
    CINEMO_LOGTARGET_SYSLOG,
    CINEMO_LOGTARGET_NULL,
    CINEMO_LOGTARGET_INVALID
} CINEMO_LOGTARGET;
```

Valid parameters for a file target are:

```
compression=gzip
rollingMaxSizeBytes=<number of bytes>
```

```
rollingMaxDurationSeconds=<number of seconds>
rollingMaxIndex=<max index>
```

E.g. with the filename "output.txt.gz?compression=gzip&rollingMaxSizeBytes=5000000" a rolling file scheme is used where a compressed log file is not bigger than 5 Megabytes.

3.10.2 Open

Open a callback target for getting log output. If a target is already opened, the current one will be closed and the new one is opened using the current settings.

Syntax

```
CinemoError Open(
    [in] void * puser,
    [in] CinemoLogCallback pcall,
);
```

Parameters

- **puser**

Argument for setting userdata passed to the callback function.

- **pcall**

Argument for setting the callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The Cinemo Media Engine generates log messages. This method gives you the ability to receive these log messages by means of a callback. Please note that the specified callback function may be called asynchronously from any internal Cinemo thread, but an internal mutex implementation prevents the callback function from being called concurrently by multiple threads at the same time. Therefore, do not call Cinemo APIs from within the *CinemoLogCallback* function as this could result in a deadlock.

3.10.3 Close

Close a target and reset all settings.

Syntax

```
CinemoError Close();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method closes the current target, frees all associated resources and resets the internal state without releasing the instance.

3.10.4 SetLevel

Set the verbosity level for all or a subset of log messages. SetLevel has to be called at least once to receive any log messages. By means of multiple calls of SetLevel with a different szfullpath the user is able to specify which log messages are output in a fine grained way.

Syntax

```
CinemoError SetLevel(
    [in] CINEMO_LOGLEVEL level,
    [in] const char * szfullpath = 0,
);
```

Parameters

- **level**

The level for filtering log message.

- **szfullpath**

Optional argument for specifying the subset of log messages for that the given level applies.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will change the level for filtering log messages coming from the Cinemo Media Engine. The szfullpath string uses a filepath syntax with a slash ("/") as separator. E.g. "/" corresponds to the root (all log messages) and "/SDK" to log messages from the SDK. All cinemo log statements are assigned to a node in a logging tree, so that it is possible to selectively receive log messages. If the szfullpath argument is given, the filtering level is applied to only those log messages that are in the respective set.

The *CINEMO_LOGLEVEL* enumeration is defined here:

```
typedef enum {
    CINEMO_LOGLEVEL_VERBOSE = -1
    CINEMO_LOGLEVEL_TRACE,
    CINEMO_LOGLEVEL_DEBUG,
    CINEMO_LOGLEVEL_INFO,
    CINEMO_LOGLEVEL_WARN,
    CINEMO_LOGLEVEL_ERROR,
    CINEMO_LOGLEVEL_FATAL,
    CINEMO_LOGLEVEL_OFF,
    CINEMO_LOGLEVEL_UNSET
} CINEMO_LOGLEVEL;
```

3.10.5 SetLayout

Set the layout of log messages.

Syntax

```
CinemoError SetLayout(  
    [in] CINEMO_LOGLAYOUT layout  
) ;
```

Parameters

- **layout**

The layout used for outputting log messages.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

The *CINEMO_LOGLAYOUT* enumeration is defined here:

```
typedef enum {  
    CINEMO_LOGLAYOUT_CLASSIC = 0,  
    CINEMO_LOGLAYOUT_MINIMAL,  
    CINEMO_LOGLAYOUT_NORMAL,  
    CINEMO_LOGLAYOUT_FULL,  
    CINEMO_LOGLAYOUT_CSV,  
    CINEMO_LOGLAYOUT_INVALID,  
} CINEMO_LOGLAYOUT;
```

3.11 ICinemoLogger

The [ICinemoLogger](#) interface provides means for emitting log messages. It can be created using the *CinemoCreateLogger()* function defined in *cinemo.h*.

ICinemoLogger Methods

METHOD	DESCRIPTION
<u>Message</u>	Emit a log message.
<u>Message</u>	Emit a log message with extended information.

3.11.1 Message

Emit a log message to be output by Cinemo.

Syntax

```
CinemoError Message(  
    [in] CINEMO_LOGLEVEL level,  
    [in] const char * szformat,  
    ...  
) ;
```

Parameters

- **level**

Log level of the message.

- **szformat**

Pointer to a UTF-8 character string with the log message.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.11.2 Message

Emit a log message with extended information to be output by Cinemo.

Syntax

```
CinemoError Message(  
    [in] CINEMO_LOGLEVEL level,  
    [in] int line,  
    [in] const char * szfile,  
    [in] const char * szfunc,  
    [in] const char * szformat,  
    ...  
) ;
```

Parameters

- **level**

Log level of the message.

- **line**

The line number where the log statement is written.

- **szfile**

Pointer to a UTF-8 character string with the filename where the log statement is written.

- **szfunc**

Pointer to a UTF-8 character string with the function name where the log statement is written.

- **szformat**

Pointer to a UTF-8 character string with the log message.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.12 ICinemoLiveStream

The [ICinemoLiveStream](#) interface provides methods for a calling application to supply Cinemo with live stream data for playback. This thread-safe interface can be obtained by calling [CinemoCreateLiveStream\(\)](#). After obtaining the interface, the live stream may be added to a playlist by passing the interface to [ICinemoPlaylist::AppendSource\(\)](#). The resulting track can be played like any other track in a playlist.

ICinemoLiveStream Methods

METHOD	DESCRIPTION
Write	Write live stream data
WriteMetadata	Write optional stream metadata
WriteDiscontinuity	Signal a stream discontinuity
WriteFormatChange	Signal that the stream format has changed
SignalServiceChange	Signal a service change and restart playback pipeline
Drain	Drain the stream
Flush	Flush the stream

Notes

The MIME type of a live stream must be specified when calling [CinemoCreateLiveStream\(\)](#). The list of supported MIME types is as follows:

MIMETYPE	DESCRIPTION
audio/pcm	Uncompressed PCM data
audio/aac	ADTS formatted AAC stream
audio/MP4A-LATM	LATM formatted AAC stream
audio/mpeg	MPEG-1/-2 audio layer 1, 2 or 3
audio/sbc	Raw SBC frames
video/MP2T	MPEG-2 Transport Stream
video/h264	H.264/MPEG-4 AVC Elementary Stream
video/x-rtpavp33	RTP Packets with Payload Type 33
application/rfb	Server to client messages as defined by RFB protocol

Please note that the support of these MIME types also depends on the set of supported codecs. Please refer to [Appendix A](#) for the list of enabled codecs in your project.

[ICinemoLiveStream](#) methods may be used as soon as the interface has been successfully created. The behaviour of the interface is the same regardless of whether or not the stream is currently being played. In case the stream is not being played by any ICinemoPlayer instance, written stream data will simply be discarded.

HUMAX Confidential

URL Parameters

It is possible to specify optional configuration parameters when creating [ICinemoLiveStream](#). Parameters are specified in the format of URL parameters and passed as the *szparam* argument to [CinemoCreateLiveStream\(\)](#). The list of supported parameters is as follows:

- ***filter_pid*** is used in combination with MPEG-2 Transport Streams and the configuration option [CINEMO_OPTION_MISC_TS_PACKET_CALLBACK](#). Identifiers for ***filter_pid***:

Identifier	Packet Content
pat	Program Association Table
pmt	Program Map Table
pcr	Program Clock Reference
bifs	ISO/IEC 14496-1 BIFS Referenced by InitialObjectDescriptor in Program Map Table
od	ISO/IEC 14496-1 Object Descriptor Referenced by InitialObjectDescriptor in Program Map Table
<pid_id>	Packets with the specified PID number

For example, appending the URL parameter “*filter_pid=pat,pmt,bifs,od,1234,1235*” will configure delivery of PAT, PMT, BIFS, OD and transport stream packets with PID 1234 and 1235 to the configured application callback function.

- ***pmt_sync*** can be set for MPEG 2 transport streams to advise Cinemo to wait for a PMT before starting the playback. This enables Cinemo to correctly evaluate and provide the information from the PMT to the application, like the languages and order of streams.

Example: `?pmt_sync=1`

- ***expose_pids*** can be set for MPEG 2 transport streams to advise Cinemo to expose the PIDs of the video/audio streams via the metapool ([CINEMO_METANAME_VIDEO_TRACK_PID](#) and [CINEMO_METANAME_AUDIO_TRACK_PID](#)). For compatibility reasons, this option is implicitly enabled if ***pmt_sync=1***

Example: `?expose_pids=1`

- ***key*** and ***riv*** are used in combination with encrypted MPEG-2 Transport Streams according to chapter 3 [HDCP Interface Independent Adaptation Specification Revision 2.1](#).

riv = initialize vector.

8 bytes, example f0f1f2f3f4f5f6f7 where ‘f0’ is the least significant byte of the vector.

key = session key XOR secret global constant.

16 bytes, example 00112233445566778899aabbccddeeff where ‘00’ is the least significant byte of the key.

Example: ?key=00112233445566778899aabbcdddeeff&riv=f0f1f2f3f4f5f6f7

- **miracast** is used to identify MPEG-2 Transport Streams that meet the 'Wi-Fi Display Technical Specification Version 1.0.0' from the Wi-Fi Alliance.

Example: ?miracast=1

- **low_latency** can be set for streams with MIME type video/h264. If low_latency mode is enabled, the video decoder will output decoded frames immediately after decoding of a frame is completed. Please note, that this mode may only be enabled, if the video stream does not contain B-frames (bidirectionally predictive-coded frames).

Example: ?low_latency=1

3.12.1 Write

Write live stream data.

Syntax

```
CinemoError Write(  
    [in]  const void * psrc,  
    [in]  int nbytes  
) ;
```

Parameters

- **psrc**

Application supplied pointer to raw stream data.

- **nbytes**

The size of the data in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The raw stream data must correspond to the MIME type specified when the [ICinemoLiveStream](#) interface is created.

3.12.2 WriteMetadata

Write optional stream metadata.

Syntax

```
CinemoError WriteMetadata(  
    [in] struct ICinemoMetapool * ppool  
) ;
```

Parameters

- **ppool**

The metadata pool.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The metadata written by this method will normally be synchronized with the live stream, that is, it will apply to all raw stream bytes written after the method successfully returns.

3.12.3 WriteDiscontinuity

Signal a stream discontinuity.

Syntax

```
CinemoError WriteDiscontinuity();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method can be used to signal that data written by the next [Write\(\)](#) call will be discontinuous with previously written data, for example, after a seek in the stream, or a recovery from signal loss.

3.12.4 WriteFormatChange

Signal that the stream format has changed.

Syntax

```
CinemoError WriteFormatChange (
    [in]  const CinemoMediaType& media
);
```

Parameters

- **media**

The stream format description.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may be used to signal a change of stream format, but only if the format is supported or allowed by the MIME type specified when the [ICinemoLiveStream](#) interface was created. For example, if the MIME type is "audio/pcm" then the stream format may be allowed to dynamically switch between various PCM input formats (and in fact will need to be specified at least once before writing any live stream data at all).

The stream format written by this method will apply to all raw stream bytes written after the method successfully returns.

3.12.5 SignalServiceChange

Signal a service change and restart playback pipeline.

Syntax

```
CinemoError SignalServiceChange();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method can be used to signal that data written by the next [Write\(\)](#) call will be discontinuous with previously written data, for example, after a change of tuner channel. All internally buffered stream data should be discarded and video renderer should be cleared.

3.12.6 Drain

Drain the live stream.

Syntax

```
CinemoError Drain();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is used to signal that the stream has ended, and that all internally buffered stream data should be played out.

3.12.7 Flush

Flush the stream.

Syntax

```
CinemoError Flush();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is used to signal that the stream has ended, and that all internally buffered stream data should be discarded.

If the stream is currently being played, then playback should stop immediately.

3.13 ICinemoProjection

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

The [ICinemoProjection](#) interface provides methods for a calling application to supply Cinemo with audio/video live elementary stream data for low-latency rendering. It can be obtained by calling [CinemoCreateProjection\(\)](#). After obtaining the interface, the object may be added to a playlist by passing the interface to [ICinemoPlaylist::AppendSource\(\)](#). The resulting track can be played like any other track in a playlist.

ICinemoProjection Methods

METHOD	DESCRIPTION
WriteVideoFormat	Write video format
WriteVideo	Write video stream data
SetAudioParams	Set audio params for stream
WriteAudioFormat	Write audio format for stream
WriteAudio	Write audio stream data
FlushAudio	Flush audio playback for stream

3.13.1 WriteVideoFormat

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Write video format.

Syntax

```
CinemoError WriteVideoFormat(  
    [in] const CinemoMediaType &media  
) ;
```

Parameters

- **media**

Application supplied media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method must be called before video elementary stream data can be supplied. In order to optimize latency, *CinemoVideoFormatFlags.low_latency* should be set in the supplied media type if the stream doesn't use frame reordering / B-frames, *CinemoVideoFormatFlags.chunked* should be set to indicate that one access unit / video frame will be provided with each call to [ICinemoProjection::WriteVideo\(\)](#). The call is only considered in players which successfully connected to the *ICinemoProjection* object (received *CINEMO_EC_OPEN* without error indication).

3.13.2 WriteVideo

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Write video stream data.

Syntax

```
CinemoError WriteVideo(  
    [in] const void * psrc,  
    [in] uint32 nbytes,  
    [in] pts70mhz npts,  
    [in] uint32 ptsvalid  
) ;
```

Parameters

- **psrc**
Pointer to video elementary stream data.
- **nbytes**
Number of bytes of video elementary stream data.
- **npts**
Presentation timestamp associated with video elementary stream data.
- **ptsvalid**
Set to non-null if supplied timestamp is valid.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Via this method the actual video elementary stream data is supplied. Currently a presentation timestamp value is not used, so it is recommended to set ptsvalid to 0. The call is only considered in players which successfully connected to the ICinemoProjection object (received CINEMO_EC_OPEN without error indication) and which are running (reception of CINEMO_EC_CUE with a non-zero cue state is a reasonable indication for this).

3.13.3 SetAudioParams

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Set audio params for stream.

Syntax

```
CinemoError SetAudioParams(  
    [in]  uint32 stream_id,  
    [in]  const CinemoAudioParams& params  
) ;
```

Parameters

- **stream_id**

Identify stream for which to set audio params.

- **params**

Audio params to set for stream.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is used to configure the audio rendering params for the specified stream. If the method is not called for a stream, default audio rendering parameters are used.

3.13.4 WriteAudioFormat

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Write audio format for stream.

Syntax

```
CinemoError WriteAudioFormat(  
    [in]  uint32 stream_id,  
    [in]  const CinemoMediaType& media  
) ;
```

Parameters

- **stream_id**

Identify stream for which to set audio format.

- **media**

Audio stream format to set.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method must be called before audio elementary stream data for the specified stream can be supplied.

3.13.5 WriteAudio

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Write audio stream data.

Syntax

```
CinemoError WriteAudio(
    [in]  uint32 stream_id,
    [in]  const void * psrc,
    [in]  uint32 nbytes,
    [in]  pts70mhz npts,
    [in]  uint32 ptsvalid
);
```

Parameters

- **stream_id**

Identify stream for which to write audio elementary stream data.

- **psrc**

Pointer to audio elementary stream data.

- **nbytes**

Number of bytes of audio elementary stream data.

- **npts**

Presentation timestamp associated with audio elementary stream data.

- **ptsvalid**

Set to non-null if supplied timestamp is valid.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Via this method the actual audio elementary stream data is supplied. Currently a presentation timestamp value is not used, so it is recommended to set ptsvalid to 0.

3.13.6 FlushAudio

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Flush audio playback for stream.

Syntax

```
CinemoError FlushAudio(  
    [in]  uint32 stream_id  
) ;
```

Parameters

- **stream_id**

Identify stream to flush.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is used to immediately stop processing and playback of the specified audio stream.

3.14 ICinemoMetapool

The [ICinemoMetapool](#) interface provides thread-safe access to metadata items. A metadata item can be, for example, the title or artist tag of an MP3 file, and is represented as a key-value pair, where the key is a generic string, and the value is of various pre-defined types.

An initially empty [ICinemoMetapool](#) instance may be created by calling [CinemoCreateMetapool\(\)](#). Otherwise, various interfaces such as [ICinemoPlayer](#), [ICinemoVFS](#), and [ICinemoPlaylist](#) will provide populated [ICinemoMetapool](#) instances.

ICinemoMetapool Methods

METHOD	DESCRIPTION
GetItemCount	Get count of available metadata items
GetItem	Get metadata item at position
GetCursor	Obtain an ICinemoMetapoolCursor interface
Read	Read metadata
Find	Find a specific metadata item
Find	Find a specific metadata item (with specified title and index)
GetText	Find and return metadata as a UTF-8 string
GetBlob	Find and return metadata as a blob
GetUint32	Find and return metadata as a 32-bit integer value
GetUint64	Find and return metadata as a 64-bit integer value
AddItem	Add a new metadata item
AddUTF8	Add a new metadata item of type UTF-8 string
AddUint32	Add a new metadata item of type 32-bit integer
AddUint64	Add a new metadata item of type 64-bit integer
AddPool	Add metadata items from another ICinemoMetapool instance
RemoveAll	Remove all metadata items
Refresh	Refresh metadata

3.14.1 GetItemCount

Return the count of available metadata items.

Syntax

```
uint32 GetItemCount() const;
```

Return value

If the method succeeds, it returns the count of available metadata items, as an unsigned 32-bit integer. Otherwise, it returns 0.

Remarks

The return value will give you the highest index that can be used when querying metadata items using the [GetItem\(\)](#) method.

3.14.2 GetItem

Return metadata attributes for an item at a specified position.

Syntax

```
CinemoError GetItem(
    [in]  uint32 position,
    [out] CinemoMetaAttributes& attrs
) const;
```

Parameters

- **position**

One-based metadata item position.

- **attrs**

For returning the attributes of the metadata item.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method gives fast random access to the attributes of all currently stored items. If the method succeeds, a *CinemoMetaAttributes* structure will be initialized with the attributes of the metadata item at specified position.

The *CinemoMetaAttributes* structure is defined here:

```
typedef struct {
    char metaname[40];           /* metadata name */
    CINEMO_METATYPE metatype;    /* metadata type */
    CINEMO_METALANG metalang;   /* metadata language code */
    uint32 id;                  /* unique id */
    uint32 title;               /* title */
    uint32 index;               /* index */
    uint32 bytes;               /* bytes of data */
} CinemoMetaAttributes;
```

The *id* field will be used as an argument to the [Read\(\)](#) method when reading the metadata corresponding to the item at specified index. If you use the [GetText\(\)](#) helper methods to return metadata strings, the *title* and *index* fields will be used as arguments.

3.14.3 GetCursor

Obtain an [ICinemoMetapoolCursor](#) interface for sequentially reading metadata attributes that match specified criteria.

Syntax

```
CinemoError GetCursor(
    [in] CINEMO_METANAME metaname,
    [in] CINEMO_METATYPE metatype,
    [in] CINEMO_METALANG metalang,
    [in] uint32 title,
    [in] uint32 index,
    [out] struct ICinemoMetapoolCursor ** pp
) const;
```

Parameters

- **metaname**

The metadata name to search for.

- **metatype**

The metadata type to search for.

- **metalang**

The metadata language to search, or *CINEMO_METALANG_UNSPECIFIED*.

- **title**

The title identifier to search, or *CINEMO_TITLE_ANY*.

- **index**

The index identifier to search, or *CINEMO_INDEX_ANY*.

- **pp**

The address of a pointer to receive the requested [ICinemoMetapoolCursor](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, it will return an [ICinemoMetapoolCursor](#) interface that provides sequential access to metadata that match the specified criteria. This interface must be released when no longer required, using the [ICinemoUnknown::Release\(\)](#) method.

The *metalang* argument may be set to *CINEMO_METALANG_UNSPECIFIED* if no specific language is required. The *title* argument may be set to *CINEMO_TITLE_ANY* if no specific title is required. The *index* argument may be set to *CINEMO_INDEX_ANY* if no specific index is required.

3.14.4 Read

Read metadata for a given ID.

Syntax

```
CinemoError Read(
    [out] void * pbits,
    [in]  uint32 id,
    [in]  uint32 nseek,
    [in]  uint32 nsize
) const;
```

Parameters

- **pbits**

Pointer to a buffer for returning the metadata.

- **id**

The metadata item ID.

- **nseek**

Number of bytes to skip from the beginning of the metadata.

- **nsize**

Size of the pbits buffer in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, the *pbits* buffer will be filled with *nsize* bytes of metadata, starting from the *nseek* offset. The *id* argument, representing the metadata item ID, has to match the value of the *id* field in the *CinemoMetaAttributes* structure associated to the item.

Cinemo internally treats metadata items similar to files. So, the *nseek* field is treated like an offset from the beginning of the metadata. By setting it to a non-zero value, you will skip the corresponding first bytes of metadata.

Please note, that metadata of type *CINEMO_METATYPE_UTF8* does not contain string termination characters. Refer to [GetText\(\)](#) as a simplified method for accessing metadata of type *CINEMO_METATYPE_UTF8*.

3.14.5 Find

Find a specific metadata item.

Syntax

```
CinemoError Find(
    [in]    CINEMO_METANAME metaname,
    [in]    CINEMO_METATYPE metatype,
    [in]    CINEMO_METALANG metalang,
    [out]   CinemoMetaAttributes& attrs
) const;
```

Parameters

- **metaname**

The metadata name to search for.

- **metatype**

The metadata type to search for.

- **metalang**

The metadata language to search, or *CINEMO_METALANG_UNSPECIFIED*.

- **attrs**

Return the attributes of the metadata item, if found.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds and finds a valid metadata item, a *CinemoMetaAttributes* structure will be initialized with its attributes.

The *metalang* argument may be set to *CINEMO_METALANG_UNSPECIFIED* if you do not care which language to search for. In this case the method will return the first metadata item matching the specified name and type, regardless of language.

3.14.6 Find

Find a specific metadata item (with specified title and index).

Syntax

```
CinemoError Find(
    [in]    CINEMO_METANAME metaname,
    [in]    CINEMO_METATYPE metatype,
    [in]    CINEMO_METALANG metalang,
    [in]    uint32 title,
    [in]    uint32 index,
    [out]   CinemoMetaAttributes& attrs
) const;
```

Parameters

- **metaname**

The metadata name to search for.

- **metatype**

The metadata type to search for.

- **metalang**

The metadata language to search, or *CINEMO_METALANG_UNSPECIFIED*.

- **title**

The title identifier to search, or *CINEMO_TITLE_ANY*.

- **index**

The index identifier to search, or *CINEMO_INDEX_ANY*.

- **attrs**

Return the attributes of the metadata item, if found.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds and finds a valid metadata item, a *CinemoMetaAttributes* structure will be initialized with its attributes.

The *metalang* argument may be set to *CINEMO_METALANG_UNSPECIFIED*, if no specific language is required. The *title* argument may be set to *CINEMO_TITLE_ANY* if no specific title is required. The *index* argument may be set to *CINEMO_INDEX_ANY* if no specific index is required. In these cases the method will return the first metadata item matching the other parameters.

3.14.7 GetText

Find and return metadata as a UTF-8 string.

Syntax

```
CinemoError GetText(
    [in] CINEMO_METANAME metaname,
    [in] CINEMO_METALANG metalang,
    [in] uint32 title,
    [in] uint32 index,
    [out] struct ICinemoUTF8 ** pp
) const;
```

Parameters

- **metaname**

The name of the metadata string.

- **metalang**

The desired language, or *CINEMO_METALANG_UNSPECIFIED*.

- **title**

The title identifier of the metadata item.

- **index**

The index of the metadata item.

- **pp**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, it will return an [ICinemoUTF8](#) interface that holds the metadata string. This interface must be released when no longer required, using the [ICinemoUnknown::Release\(\)](#) method. If no metadata items are found matching the specified criteria, the method will return *CinemoErrorNoSuchObject*.

If for the specified metaname and metalang no item with metatype *CINEMO_METATYPE_UTF8* exists, items of metatypes *CINEMO_METATYPE_UINT32* or *CINEMO_METATYPE_UIN64* might be converted and returned instead.

The *metalang* argument may be set to *CINEMO_METALANG_UNSPECIFIED*, if no specific language is required. The *title* argument may be set to *CINEMO_TITLE_ANY* if no specific title is required. The *index* argument may be set to *CINEMO_INDEX_ANY* if no specific index is required. In these cases the method will return the first metadata string matching the other parameters.

3.14.8 GetBlob

Find and return metadata as a blob.

Syntax

```
CinemoError GetBlob(
    [in]  CINEMO_METANAME metaname,
    [in]  CINEMO_METATYPE metatype,
    [in]  uint32 title,
    [in]  uint32 index,
    [out] struct ICinemoBlob ** pp
) const;
```

Parameters

- **metaname**

The name of the metadata string.

- **metatype**

The type of the metadata item.

- **title**

The title identifier of the metadata item.

- **index**

The index of the metadata item.

- **pp**

The address of a pointer to receive the requested [ICinemoBlob](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, it will return an [ICinemoBlob](#) interface that holds the metadata string. This interface must be released when no longer required, using the [ICinemoUnknown::Release\(\)](#) method. If no metadata items are found matching the specified criteria, the method will return *CinemoErrorNoSuchObject*.

The *title* argument may be set to *CINEMO_TITLE_ANY* if no specific title is required. The *index* argument may be set to *CINEMO_INDEX_ANY* if no specific index is required. In these cases the method will return the first metadata string matching the other parameters.

3.14.9 GetUint32

3.14.10 GetUint64

Find and return metadata as a 32-bit or 64-bit integer value.

Syntax

```
CinemoError GetUint32(
    [in] CINEMO_METANAME metaname,
    [in] uint32 title,
    [in] uint32 index,
    [out] uint32 * p
) const;

CinemoError GetUint64(
    [in] CINEMO_METANAME metaname,
    [in] uint32 title,
    [in] uint32 index,
    [out] uint64 * p
) const;
```

Parameters

- **metaname**

The name of the metadata string.

- **title**

The title identifier of the metadata item.

- **index**

The index of the metadata item.

- **p**

The address of an integer value to receive the metadata.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, it will return the specified 32-bit or 64-bit value. If no metadata items are found matching the specified criteria, the method will return *CinemoErrorNoSuchObject*.

Please note that [GetUint32\(\)](#) will not return items which are registered as *CINEMO_METATYPE_UINT64* and vice versa.

The *title* argument may be set to *CINEMO_TITLE_ANY* if no specific title is required. The *index* argument may be set to *CINEMO_INDEX_ANY* if no specific index is required. In these cases the method will return the first metadata string matching the other parameters.

3.14.11 AddItem

Add a metadata item to the pool.

Syntax

```
CinemoError AddItem(
    [in] CINEMO_METANAME metaname,
    [in] CINEMO_METATYPE metatype,
    [in] CINEMO_METALANG metalang,
    [in] uint32 title,
    [in] uint32 index,
    [in] const void * pbits,
    [in] uint32 bytes,
    [in] uint32 flags = 0
);
```

Parameters

- **metaname**

The name of the metadata item.

- **metatype**

The type of the metadata item.

- **metalang**

The language of the metadata item (or *CINEMO_METALANG_UNSPECIFIED*).

If metatype is other than *CINEMO_METATYPE_UTF8*, set to *CINEMO_METALANG_NONE*.

- **title**

The title identifier of the metadata item. Do not set to *CINEMO_TITLE_ANY*.

- **index**

The index of the metadata item. Do not set to *CINEMO_INDEX_ANY*.

- **pbits**

Pointer to the metadata. The memory will be copied.

- **bytes**

The size of the metadata item, in bytes.

- **flags (optional argument)**

CINEMO_METAPOOLFLAG bitmask.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method allows an application to supply metadata items. For example, additional metadata may be written for each track in an [ICinemoPlaylist](#), which may be used to provide additional information for Cinemo's

distributed playback discovery service. Many other Cinemo API methods make use of metadata provided by the application in this way. The optional flags arguments specifies properties for accessing the metapool and adding items.

- **CINEMO_METAPOOLFLAG_NONE**

No flags, i.e. default behaviour.

- **CINEMO_METAPOOLFLAG_ALLOW_EMPTY_DATA**

Adds an item although the payload is empty else eventually removes existing item but does not add a new one.

- **CINEMO_METAPOOLFLAG_NO_OVERWRITE**

Returns an error if an item with same attributes already exist.

- **CINEMO_METAPOOLFLAG_INCREMENT_INDEX**

Automatically increments the index by the number of existing items with same metaname.

- **CINEMO_METAPOOLFLAG_UNIQUE_INDEX**

Does not add an item if (name/title/type/lang/payload) already exist for any index else increments the index by the maximum index of an existing item with same (name/title/type/lang).

3.14.12 AddUTF8

Add a metadata item of type UTF-8 string to the pool.

Syntax

```
CinemoError AddUTF8 (
    [in]  CINEMO_METANAME metaname,
    [in]  CINEMO_METALANG metalang,
    [in]  uint32 title,
    [in]  uint32 index,
    [in]  const char * szutf8,
    [in]  uint32 flags = 0
);
```

Parameters

- **metaname**

The name of the metadata item.

- **metalang**

The language of the metadata item (or *CINEMO_METALANG_UNSPECIFIED*).

- **title**

The title identifier of the metadata item.

- **index**

The index of the metadata item.

- **szutf8**

Pointer to a zero terminated UTF-8 character string.

- **flags (optional argument)**

CINEMO_METAPOOLFLAG bitmask.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method allows an application to supply metadata items of type UTF-8 string.

3.14.13 AddUInt32

3.14.14 AddUInt64

Add a metadata item of type 32-bit or 64-bit integer to the pool.

Syntax

```
CinemoError AddUInt32(
    [in] CINEMO_METANAME metaname,
    [in] uint32 title,
    [in] uint32 index,
    [in] uint32 value,
    [in] uint32 flags = 0
);

CinemoError AddUInt64(
    [in] CINEMO_METANAME metaname,
    [in] uint32 title,
    [in] uint32 index,
    [in] uint64 value,
    [in] uint32 flags = 0
);
```

Parameters

- **metaname**

The name of the metadata item.

- **title**

The title identifier of the metadata item.

- **index**

The index of the metadata item.

- **value**

The integer value.

- **flags (optional argument)**

CINEMO_METAPOOLFLAG bitmask.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method allows an application to supply metadata items of type 32-bit or 64-bit integer.

3.14.15 AddPool

Add all metadata items from another [ICinemoMetapool](#).

Syntax

```
CinemoError AddPool(  
    [in]  const struct ICinemoMetapool * ppool  
) ;
```

Parameters

- **ppool**

The [ICinemoMetapool](#) containing metadata items to add.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method allows an application to duplicate the contents of a metapool.

3.14.16 RemoveAll

Remove all metadata items from the pool.

Syntax

```
CinemoError RemoveAll();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.14.17 Refresh

Refresh metadata, delete all old metadata.

Syntax

```
CinemoError Refresh();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method should only be called after receiving *CINEMO_EC_METADATA* event, if you have already obtained an [ICinemoMetapool](#) interface from an [ICinemoPlayer](#) instance. This method will update the current metadata items from the running player instance. The reason is to avoid synchronization errors with the calling application, since metadata may otherwise be updated asynchronously by the running [ICinemoPlayer](#) instance.

3.15 ICinemoMetapoolCursor

The [ICinemoMetapoolCursor](#) interface provides thread-safe sequential access to metadata.

ICinemoMetapoolCursor Methods

METHOD	DESCRIPTION
GetNextItem	Get metadata attributes of the next item

3.15.1 GetNextItem

Get metadata attributes of the next item, i.e. the first call returns attributes of the first item, the second call of the next one etc.

Syntax

```
CinemoError GetNextItem(  
    [out] CinemoMetaAttributes& attrs  
) ;
```

Parameters

- **attrs**

Return the attributes of the metadata item

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a *CinemoMetaAttributes* structure will be initialized with its attributes and the internal item pointer is advanced so that the next call will return the attributes of the next item.

3.16 ICinemoOption

The [ICinemoOption](#) interface provides methods to assign, retrieve, load and save Cinemo Media Engine options. The interface may be obtained in one of the following three ways:

- `CinemoCreateConfig()`
- `ICinemoPlayer::QueryInterface(ICinemoOptions::iid)`
- `ICinemoVFS::QueryInterface(ICinemoOptions::iid)`

In the first case, the returned [ICinemoOption](#) interface will access Cinemo *global* options. Any options set here will be applied to all [ICinemoPlayer](#) and all [ICinemoVFS](#) instances subsequently created, but not to any instances created before this. In the second case, the returned [ICinemoOption](#) interface will access options local to the player or VFS instance. Any options set here will apply only to the player or VFS instance. This interface is thread-safe when considering the remarks in the [Notes](#) section.

ICinemoOption Methods

METHOD	DESCRIPTION
SetOption	Set option value
GetOption	Get option value
GetOptionType	Get option type
GetOptionFlag	Get option flags
GetOptionEnum	Get option enumerated types
GetOptionID	Get option ID
SaveOptions	Save options to XML file
LoadOptions	Load options from XML file
ResetOptions	Reset all options to default values
SetOptionBlob	Set option blob value
GetOptionBlob	Get option blob value
SetOptionCallback	Set option callback function
GetOptionCallback	Get option callback function

Notes

The option system assumes a static adjustment of options by the caller: Player or VFS specific adjustments of options shall be applied after obtaining the respective interface, and before issuing the start of playback or metadata extraction. Other (dynamic) adjustments to options shall only be made after consulting back with Cinemo.

3.16.1 SetOption

Set option with the given ID to the given value.

Syntax

```
CinemoError SetOption(  
    [in] const char * szid,  
    [in] const char * szvalue  
) ;
```

Parameters

- **szid**

The option ID.

- **szvalue**

The new value to set the option to.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The recommended approach to get the ID of a specific option is the use the [GetOptionID\(\)](#) method inside a loop incrementing the index, until you get an error. But there is also a set of *CINEMO_OPTION* strings that can be used for programmatically hard-coding Cinemo Media Engine options.

3.16.2 GetOption

Get the value of the option with the given ID.

Syntax

```
CinemoError GetOption(  
    [in]  const char * szid,  
    [out] struct ICinemoUTF8 ** pp  
) ;
```

Parameters

- **szid**

The option ID.

- **pp**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, the pp will point to an [ICinemoUTF8](#) instance that will hold the option value as a string. This instance has to be manually released afterwards, using the Release method. The recommended approach to get the ID of a specific option is the use the [GetOptionID\(\)](#) method inside a loop incrementing the index, until you get an error. But there is also a set of *CINEMO_OPTION* strings that can be used for programmatically hard-coding Cinemo Media Engine options.

3.16.3 GetOptionType

Get the type of the option with the given ID.

Syntax

```
CinemoError GetOptionType(
    [in] const char * szid,
    [out] CINEMO_OPTIONTYPE& type
);
```

Parameters

- **szid**

The option ID.

- **type**

Returns the option type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a value from the *CINEMO_OPTIONTYPE* enumeration will be set in the *type* argument.

The *CINEMO_OPTIONTYPE* enumeration is defined here:

```
typedef enum {
    CINEMO_OPTIONTYPE_NONE,                                /* invalid */
    CINEMO_OPTIONTYPE_BOOL,                               /* boolean value */
    CINEMO_OPTIONTYPE_INTEGER,                            /* integer value */
    CINEMO_OPTIONTYPE_MS,                                /* integer milliseconds */
    CINEMO_OPTIONTYPE_SECONDS,                           /* integer seconds */
    CINEMO_OPTIONTYPE_HZ,                                /* integer Hz */
    CINEMO_OPTIONTYPE_BYTES,                            /* integer bytes */
    CINEMO_OPTIONTYPE_KB,                                /* integer KB */
    CINEMO_OPTIONTYPE_MB,                                /* integer MB */
    CINEMO_OPTIONTYPE_GB,                                /* integer GB */
    CINEMO_OPTIONTYPE_SECTORS,                           /* integer sectors */
    CINEMO_OPTIONTYPE_PIXELS,                           /* integer pixels */
    CINEMO_OPTIONTYPE_ENUM,                             /* enumerated type */
    CINEMO_OPTIONTYPE_TEXT,                            /* UTF-8 text */
    CINEMO_OPTIONTYPE_REGEX,                           /* UTF-8 regular expression */
    CINEMO_OPTIONTYPE_BLOB,                            /* memory blob */
    CINEMO_OPTIONTYPE_CALLBACK                         /* callback function pointer with user
context */
} CINEMO_OPTIONTYPE;
```

The recommended approach to get the ID of a specific option is the use the [GetOptionID\(\)](#) method inside a loop incrementing the index, until you get an error. But there is also a set of *CINEMO_OPTION* strings that can be used for programmatically hard-coding Cinemo Media Engine options.

HUMAX AUTOMOTIVE
CONFIDENTIAL

3.16.4 GetOptionFlag

Get the flags of the option with the given ID.

Syntax

```
CinemoError GetOptionFlag(
    [in] const char * szid,
    [out] CINEMO_OPTIONFLAG& flag
);
```

Parameters

- **szid**

The option ID.

- **flags**

Returns the flags of the option.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a value from the *CINEMO_OPTIONFLAG* enumeration will be set in the *flag* argument.

The *CINEMO_OPTIONFLAG* enumeration, which defines flags associated with an option, is defined here:

```
typedef enum {
    CINEMO_OPTIONFLAG_NONE          = 0,
    CINEMO_OPTIONFLAG_HIDDEN        = 1 << 0,
    CINEMO_OPTIONFLAG_DONT_SAVE_EMPTY = 1 << 1,
    CINEMO_OPTIONFLAG_DONT_SAVE    = 1 << 2,
    CINEMO_OPTIONFLAG_CONST        = 1 << 3,
    CINEMO_OPTIONFLAG_GLOBAL       = 1 << 4,
    CINEMO_OPTIONFLAG_NIC          = 1 << 5,
    CINEMO_OPTIONFLAG_STRING       = 1 << 6,
} CINEMO_OPTIONFLAG;
```

The recommended approach to get the ID of a specific option is the use the [GetOptionID\(\)](#) method inside a loop incrementing the index, until you get an error. But there is also a set of *CINEMO_OPTION* strings that can be used for programmatically hard-coding Cinemo Media Engine options.

3.16.5 GetOptionEnum

Return a string with all possible enumerated types of the option with the given ID.

Syntax

```
CinemoError GetOptionEnum(  
    [in]  const char * szid,  
    [out] struct ICinemoUTF8 ** pp  
) ;
```

Parameters

- **szid**

The option ID.

- **pp**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, *pp* will point to an [ICinemoUTF8](#) instance that will hold the string containing all possible enumerated types of the option. This instance has to be manually released afterwards, using the *Release* method.

The recommended approach to get the ID of a specific option is the use the [GetOptionID\(\)](#) method inside a loop incrementing the index, until you get an error. But there is also a set of *CINEMO_OPTION* strings that can be used for programmatically hard-coding Cinemo Media Engine options.

3.16.6 GetOptionID

Return the option ID at a specified index.

Syntax

```
CinemoError GetOptionID(  
    [in]  uint32 index,  
    [out] struct ICinemoUTF8 ** pp  
) ;
```

Parameters

- **index**

The option index.

- **pp**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, *pp* will point to an [ICinemoUTF8](#) instance that will hold the string containing the option ID. This instance has to be manually released afterwards, using the *Release* method.

The options indexes are zero-based, so the first option index will be 0. To enumerate all the options IDs, simply loop over the indexes, starting at 0, until the [GetOptionID\(\)](#) returns an error.

3.16.7 SaveOptions

Save options to an XML file.

Syntax

```
CinemoError SaveOptions(  
    [in] const char * szfilename  
) ;
```

Parameters

- **szfilename**

The path to the XML file.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method provides a way to archive the options values to an XML file. This file can afterwards be un-archived using the [LoadOptions\(\)](#) method.

3.16.8 LoadOptions

Load options from an XML file.

Syntax

```
CinemoError LoadOptions(  
    [in] const char * szfilename  
) ;
```

Parameters

- **szfilename**

The path to the XML file.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method provides a way to un-archive previously saved options values inside an XML file using the [SaveOptions\(\)](#) method.

3.16.9 ResetOptions

Reset all options to their default values.

Syntax

```
CinemoError ResetOptions();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.16.10 SetOptionBlob

Set option blob with the given ID to the given value.

Syntax

```
CinemoError SetOptionBlob(  
    [in]    const char * szid,  
    [in]    const void * pdata,  
    [in]    uint32 nbytes  
) ;
```

Parameters

- **szid**

The option ID.

- **pdata**

The new value to set the option to.

- **nbytes**

The size of the new value.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

There are currently no options that use this interface.

3.16.11 GetOptionBlob

Get the value of the option blob with the given ID.

Syntax

```
CinemoError GetOptionBlob(  
    [in]  const char * szid,  
    [out] struct ICinemoBlob ** pp  
) ;
```

Parameters

- **szid**

The option ID.

- **pp**

The address of a pointer to receive the requested [ICinemoBlob](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, the *pp* will point to an [ICinemoBlob](#) instance that will hold the option value. This instance has to be manually released afterwards, using the [ICinemoUnknown::Release\(\)](#) method. There are currently no options that use this interface.

3.16.12 SetOptionCallback

Set the option with the given ID to the given callback function.

Syntax

```
CinemoError SetOptionCallback (
    [in]     const char * szid,
    [in]     const void * pcall,
    [in]     void * puser
);
```

Parameters

- **szid**

The option ID.

- **pcall**

Pointer to the callback function.

- **puser**

User-defined parameter which will be passed to the callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.16.13 GetOptionCallback

Get the value of the option blob with the given ID.

Syntax

```
CinemoError GetOptionCallback (
    [in]     const char * szid,
    [out]    const void *& pcall,
    [out]    void *& puser
);
```

Parameters

- **szid**

The option ID.

- **pcall**

Pointer to the callback function.

- **puser**

User-defined parameter which is set to be passed to the callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.17 ICinemoConfig

The [ICinemoConfig](#) interface provides methods to set the Cinemo Media Engine configuration. This is typically required before calling methods of any other interfaces. This thread-safe interface is obtained by calling [CinemoCreateConfig\(\)](#) function.

Note that [ICinemoConfig](#) interface is derived from [ICinemoOption](#). This interface provides access to the Cinemo Media Engine **global** options.

ICinemoConfig Methods

METHOD	DESCRIPTION
GetVersion	Get the current Cinemo version
GetDefaultFolders	Get the default Cinemo folder pathnames for options and plugins
SetLog	Set Cinemo log output
SetLogLevel	Set Cinemo log level
SetPluginDirectory	Set Cinemo plugin directory path
SetCustomDecoderFactory	Set factory function for creating user defined decoder objects
RefreshNetworkInterfaces	Notify the framework about a change of the network interfaces

3.17.1 GetVersion

Returns the current Cinemo Media Engine version.

Syntax

```
CinemoError GetVersion(  
    [out] CINEMO_VERSION * p  
) ;
```

Parameters

- **p**

Returns the Cinemo Media Engine version.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a *CINEMO_VERSION* structure will be initialized with the Cinemo Media Engine version.

The *CINEMO_VERSION* structure is defined here:

```
typedef struct {  
    uint32 major;           /* major version */  
    uint32 minor;           /* minor version */  
    uint32 revision;        /* revision */  
    uint32 build;           /* current build number */  
    const char* friendly_name; /* friendly name of build */  
} CINEMO_VERSION;
```

3.17.2 GetDefaultFolders

Get the default Cinemo folder pathnames for options and plugins.

Syntax

```
CinemoError GetDefaultFolders (
    [in] const char * szapp,
    [out] struct ICinemoUTF8 ** ppplugins,
    [out] struct ICinemoUTF8 ** poptions
);
```

Parameters

- **szapp**

The full path of the calling application.

- **ppplugins**

The address of a pointer to receive the default plugin folder path.

- **poptions**

The address of a pointer to receive the default options folder path.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The full path of the calling application is typically the *argv[0]* parameter passed to the application's [main\(\)](#) function. This value may be passed directly to *GetDefaultFolders()*.

3.17.3 SetLog

Specify the Cinemo Media Engine log output.

Syntax

```
CinemoError SetLog(
    [in] CINEMO_LOGTARGET target,
    [in] CINEMO_LOGLEVEL level,
    [in] const char * szfilename = 0,
    [in] CinemoLogCallback pcall = 0,
    [in] void * puser = 0
);
```

Parameters

- **target**

The target for writing log message.

- **level**

The level for filtering log message.

- **szfilename**

Optional argument for setting the path to the log file, if the target argument is set to *CINEMO_LOGTARGET_FILE* or *CINEMO_LOGTARGET_PIPE*.

- **pcall**

Optional argument for setting the callback function, if the target argument is set to *CINEMO_LOGTARGET_CALLBACK*.

- **puser**

Optional argument for setting userdata passed to the callback function, if the target argument is set to *CINEMO_LOGTARGET_CALLBACK*.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The Cinemo Media Engine outputs log messages. This method gives you the ability to control the destination of these log messages and the verbosity of the engine.

If the target argument is set to *CINEMO_LOGTARGET_CALLBACK*, please note that the specified callback function may be called asynchronously from any internal Cinemo thread, but an internal mutex implementation prevents the callback function from being called concurrently by multiple threads at the same time. Therefore, do not call Cinemo APIs from within the *CinemoLogCallback* function as this could result in a deadlock.

The *CINEMO_LOGTARGET* enumeration is defined here:

```
typedef enum {
    CINEMO_LOGTARGET_NONE = 0,
    CINEMO_LOGTARGET_FILE,
    CINEMO_LOGTARGET_PIPE,
    CINEMO_LOGTARGET_STDOUT,
    CINEMO_LOGTARGET_STDERR,
    CINEMO_LOGTARGET_CALLBACK,
    CINEMO_LOGTARGET_OUTPUT_DEBUG_STRING,
    CINEMO_LOGTARGET_SYSLOG,
} CINEMO_LOGTARGET;
```

The *CINEMO_LOGLEVEL* enumeration is defined here:

```
typedef enum {
    CINEMO_LOGLEVEL_VERBOSE = -1
    CINEMO_LOGLEVEL_TRACE,
    CINEMO_LOGLEVEL_DEBUG,
    CINEMO_LOGLEVEL_INFO,
    CINEMO_LOGLEVEL_WARN,
    CINEMO_LOGLEVEL_ERROR,
    CINEMO_LOGLEVEL_FATAL,
} CINEMO_LOGLEVEL;
```

3.17.4 SetLogLevel

Set log level without changing other log parameters.

Syntax

```
CinemoError SetLogLevel(  
    [in] CINEMO_LOGLEVEL level  
) ;
```

Parameters

- **level**

The level for filtering log message.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will change the level for filtering log messages coming from the Cinemo Media Engine. Please see the [SetLog\(\)](#) method documentation for a complete overview of the *CINEMO_LOGLEVEL* enumeration.

3.17.5 SetPluginDirectory

Specify the directory where Cinemo Media Engine plugins are located.

Syntax

```
CinemoError SetPluginDirectory(  
    [in] const char * szdir  
) ;
```

Parameters

- **szdir**

The path to the directory containing the plugins.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The Cinemo Media Engine is delivered with a complete set of plugins to handle media formats and audio/video devices. During initialization, it is your responsibility to tell the Cinemo Media Engine where these plugins are located.

3.17.6 SetCustomDecoderFactory

Specify factory function for creating user defined decoder objects.

Syntax

```
CinemoError SetCustomDecoderFactory(  
    [in] CinemoFnCreateCustomDecoder fncreate  
) ;
```

Parameters

- **fncreate**

The function pointer to the decoder factory function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Allows the usage of user defined decoders inside Cinemo. Currently the functionality is only available for dedicated audio decoders and with dedicated builds of Cinemo. For this purpose, the decoder supplied object must support the [ICinemoAudioCodec](#) interface. At run time, Cinemo will supply exactly one audio frame to the Write() function of [ICinemoAudioCodec](#). This frame needs to be decoded to PCM and delivered back to Cinemo by means of the callback function supplied during initialization of the [ICinemoAudioCodec](#) object. The sample application SamplePlayer_CustomDecoder provides a template for the implementation of a custom audio decoder.

3.17.7 RefreshNetworkInterfaces

Notify the framework about a change of the network interfaces.

Syntax

```
CinemoError RefreshNetworkInterfaces();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Used to inform the framework that the network interfaces changed and e.g. a running SSDP server has to be updated.

3.18 ICinemoPlaylist

The [ICinemoPlaylist](#) interface is used to manage playlists. Playlists may contain zero or more tracks; no tracks can be played unless they are in a playlist. This thread-safe interface is obtained by calling the [CinemoCreatePlaylist\(\)](#) function.

ICinemoPlaylist Methods

METHOD	DESCRIPTION
SetEventCallback	Set callback function to receive playlist events
SetEventQueue	Set event queue to receive playlist events
SetEventSourceID	Set source ID for posted playlist events
SetName	Set user-friendly name of the playlist
SetRepeat	Set repeat mode of the playlist
SetOrder	Set play order of the playlist
SetRandomSeed	Set random seed value for random/shuffle playlist ordering
GetName	Get user-friendly name of the playlist
GetOpenURL	Get the playlist URL which was used in last call to Open()
GetRepeat	Get repeat mode
GetOrder	Get play order
GetCount	Get number of tracks in the playlist
GetIndex	Get index of a track
GetTrack	Get track identifier at a given index
GetIndexOfTrackOffset	Get index of track at original (unshuffled) offset within playlist
GetTrackOffsetAtIndex	Get original (unshuffled) offset within playlist of track at index
Open	Open playlist from a URL
Close	Close playlist and remove all tracks
Cancel	Cancel playlist creation or other blocking calls
Enable	Enable playlist creation or other blocking calls after Cancel() .
AppendTrack	Append a track to the playlist
AppendTrackWithID	Append a track to the playlist with a user defined unique ID
AppendSource	Append a source object to the playlist
Remove	Remove tracks with given identifiers from the playlist
RemoveAll	Remove all tracks
Move	Move tracks to a specified index
Find	Get track identifier of a given URL if it exists in the playlist
GetMetapool	Get metapool for a track
GetURL	Get URL of a track

METHOD	DESCRIPTION
Select	Select the entire playlist for playback on the underlying device
SelectTrack	Select a single track for playback on the underlying device
InitPlayback	Get ICinemoPlayer interface for playback
InitPlayback2	Get ICinemoPlayer2 interface for playback
GetAlphaIndexCount	Get count of available alphabetical index items
GetAlphaIndex	Get alphabetical index item
GetCandidateCount	Get count of available next character candidates
GetCandidate	Get next character candidate
Freeze	Set freeze state of the playlist
AttachPlayer	Attach an existing ICinemoPlayer to this playlist
AttachPlayer2	Attach an existing ICinemoPlayer2 to this playlist

Notes

Playlists may be manually constructed by calling [AppendTrack\(\)](#) method for each URL to be included in the playlist. This is the simplest kind of playlist. Otherwise, playlists may be automatically constructed by calling the [Open\(\)](#) method and passing the URL of a *container* object, for example a file system folder, or a CD device (e.g. “/dev/scd0”). In this case, the playlist will be automatically filled with all immediate children of the container, and will also take care of automatically updating itself if the contents of the container are dynamically changed.

Important Note

If the [Open\(\)](#) method is called with *CINEMO_PLFLAGS_WAIT* flag, the method is *synchronous* and *blocking* – the calling application is responsible for creating threads or other synchronization objects in order to avoid, for example, hanging up the GUI while calling this method. Depending on the container object being opened, the [Open\(\)](#) method may take significant time: for example, opening a CD device may require spinning up the disc or waiting on ATAPI timeouts for scratched discs. Opening other types of container may require HTTP requests to remote servers. It cannot be emphasized enough that the correct use of [Open\(\)](#) method requires the calling application to maintain threads and implement synchronization strategies to avoid hang ups.

Any blocking method for which *CINEMO_PLFLAGS_WAIT* flag is specified may be unblocked by calling [ICinemoPlaylist::Cancel\(\)](#). In this case, the blocked method will return immediately with *CinemoErrorCancel* error code.

3.18.1 SetEventCallback

Set callback function to receive playlist events.

Syntax

```
CinemoError SetEventCallback(  
    [in] void * puser,  
    [in] CinemoEventCallback pcall  
) ;
```

Parameters

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

The application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The user defined application callback function will be called asynchronously whenever a new playlist event is available. The application is responsible for synchronizing events from multiple threads to its own application thread.

Currently the following events are supported for ICinemoPlaylist:

- [CINEMO_EC_PLAYLIST_CHANGED](#)
- [CINEMO_EC_PLAYLIST_REPEAT](#)
- [CINEMO_EC_PLAYLIST_ORDER](#)
- [CINEMO_EC_PLAYLIST_EMPTY](#)
- [CINEMO_EC_CLOSE](#)
- [CINEMO_EC_ERROR](#)
- [CINEMO_EC_SELECT](#)

Please refer to the event descriptions for more information.

To stop receiving events, a NULL callback function may be specified.

3.18.2 SetEventQueue

Set event queue to receive playlist events.

Syntax

```
CinemoError SetEventQueue(  
    [in]     struct ICinemoEventQueue * pqueue  
) ;
```

Parameters

- **queue**
pointer to [ICinemoEventQueue](#) object to receive playlist events.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This approach helps transferring the events to the application main thread using an event queue object. It behaves the same as if an application callback would push the new events to the event queue. The queue object will be kept referenced by the playlist until it is set again or the playlist is destroyed.

To stop associating a queue with the playlist, a NULL pointer may be specified.

3.18.3 SetEventSourceID

Set source ID for posted events.

Syntax

```
CinemoError SetEventSourceID(  
    [in]      uint32 source_id  
) ;
```

Parameters

- **source_id**

User-defined number passed in *CinemoEvent* *source_id* member.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This approach helps differentiate sources of events when using shared event queue object.

3.18.4 SetName

Set user-friendly name of the playlist.

Syntax

```
CinemoError SetName(  
    [in]  const char * szname  
) ;
```

Parameters

- **szname**
UTF-8 zero-terminated string name of the playlist

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.18.5 SetRepeat

Set repeat mode of the playlist.

Syntax

```
CinemoError SetRepeat(  
    [in] CINEMO_PLREPEAT repeat  
) ;
```

Parameters

- **repeat**

The playlist repeat mode

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The repeat mode defines the playback behavior when a track has finished playing. It may be set to one of the following values:

- **CINEMO_PLREPEAT_OFF**

No tracks are repeated.

- **CINEMO_PLREPEAT_ALL**

When all tracks in the playlist are finished, repeat the entire playlist.

- **CINEMO_PLREPEAT_SINGLE**

Repeat the same track.

- **CINEMO_PLREPEAT_GROUP**

Repeat group of tracks. Only supported by iAP and BT remote devices.

The default value, when a playlist is created, is *CINEMO_PLREPEAT_OFF*.

When using this method for iAP-based playlists, it will be fully asynchronous and return only the result of sending the command to the device and not the corresponding response from the device. Any state change related to repeat will be reported asynchronously as an event.

3.18.6 SetOrder

Set play order of the playlist.

Syntax

```
CinemoError SetOrder(
    [in] CINEMO_PLORDER order
);
```

Parameters

- **order**

The play order

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The play order defines the playback behavior when a track has finished playing. It may be set to one of the following values:

- **CINEMO_PLORDER_OFF**

Playback stops at the end of the track.

- **CINEMO_PLORDER_SEQUENTIAL**

Sequential play order.

- **CINEMO_PLORDER_RANDOM**

Random play order.

- **CINEMO_PLORDER_SHUFFLE**

Shuffled play order.

- **CINEMO_PLORDER_RANDOM_GROUP**

Random play order of group of tracks. Only supported by iAP and BT remote devices.

If the play order is *CINEMO_PLORDER_SEQUENTIAL*, all tracks in the playlist will be played sequentially according to their index in the playlist. [ICinemoPlaylist](#) exposes methods such as [Move\(\)](#) which enable applications to modify the index of each track, and therefore affect the sequential play order. After the last track index in the playlist has been played, the subsequent playback behavior is defined by the [SetRepeat\(\)](#) method.

If the play order is *CINEMO_PLORDER_RANDOM*, then tracks in the playlist will be played in random order. If there is more than one track in the playlist, then care is taken to not play the same track twice in a row, otherwise, the play order is completely random. The random order may be reproducible by calling [SetRandomSeed\(\)](#) before calling [SetOrder\(\)](#).

If the play order is *CINEMO_PLORDER_SHUFFLE*, then all tracks in the playlist will be shuffled and re-indexed according to their new position in the playlist. The new play order should be displayed in the application user

interface. The tracks are then played sequentially in shuffled order. The shuffled order may be reproducible by calling [SetRandomSeed\(\)](#) before calling [SetOrder\(\)](#).

The following diagram shows an example playlist with ten tracks, before and after calling [SetOrder\(\)](#) with order *CINEMO_PLORDER_SHUFFLE*:

Index	Track ID		Index	Track ID
1	1		1	4
2	2		2	2
3	3		3	5
4	4	SetOrder() SHUFFLE →	4	10
5	5		5	1
6	6		6	7
7	7		7	9
8	8		8	6
9	9		9	3
10	10		10	8

The original track order may be restored by calling [SetOrder\(\)](#) with *CINEMO_PLORDER_SEQUENTIAL*, which undoes the shuffle:

Index	Track ID		Index	Track ID
1	4		1	1
2	2		2	2
3	5		3	3
4	10	SetOrder() SEQUENTIAL →	4	4
5	1		5	5
6	7		6	6
7	9		7	7
8	6		8	8
9	3		9	9
10	8		10	10

When using this method for iAP-based playlists, it will be fully asynchronous and return only the result of sending the command to the device and not the corresponding response from the device. Any state change related to the playback order will be reported asynchronously as an event.

3.18.7 SetRandomSeed

Set random seed value for random/shuffle playlist ordering.

Syntax

```
CinemoError SetRandomSeed (
    [in]  uint32 nseed
);
```

Parameters

- **nseed**

The random seed value.

Return value

This method always returns *CinemoNoError*.

Remarks

The random seed value will be stored in the playlist, and non-zero values will be applied only when the following calls are made:

- SetOrder(CINEMO_PLORDER_RANDOM)
- SetOrder(CINEMO_PLORDER_SHUFFLE)

The random seed value is not normally assigned. It enables random or shuffled playlist orders to be reproducible, which may be useful during testing or debugging. For example, if an identical random seed value is assigned immediately before shuffling the playlist, the resulting shuffled order will be identical each time.

3.18.8 GetName

Get user-friendly name of the playlist.

Syntax

```
CinemoError GetName(  
    [out] struct ICinemoUTF8 ** pp  
) ;
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method allocates an [ICinemoUTF8](#) string interface, assigns to it the user-friendly name of the playlist previously set by calling the [SetName\(\)](#) method, and returns a pointer to the interface. It is the application's responsibility to release the interface when it is no longer required.

3.18.9 GetOpenURL

Get the playlist URL which was used in last call to [Open\(\)](#).

Syntax

```
CinemoError GetOpenURL(  
    [out] struct ICinemoUTF8 ** pp  
) ;
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method allocates an [ICinemoUTF8](#) string interface, assigns to it the URL of the playlist previously set by calling the [Open\(\)](#) method, and returns a pointer to the interface. It is the application's responsibility to release the interface when it is no longer required.

3.18.10 GetRepeat

Get repeat mode.

Syntax

```
CINEMO_PLREPEAT GetRepeat();
```

Return value

Returns the *CINEMO_PLREPEAT* mode of the playlist.

Remarks

This method returns the *CINEMO_PLREPEAT* mode set by a previous call to [SetRepeat\(\)](#).

3.18.11 GetOrder

Get play order.

Syntax

```
CINEMO_PLORDER GetOrder();
```

Return value

Returns the *CINEMO_PLORDER* of the playlist.

Remarks

This method returns the *CINEMO_PLORDER* set by a previous call to [SetOrder\(\)](#).

3.18.12 GetCount

Get number of tracks in the playlist.

Syntax

```
uint32 GetCount();
```

Return value

Return the count of tracks in the playlist, or zero if the playlist is empty.

Remarks

The count of items in a playlist may only be changed by calling [AppendTrack\(\)](#), [Remove\(\)](#) or [RemoveAll\(\)](#) methods of the playlist.

3.18.13 GetIndex

Get index of a track.

Syntax

```
uint32 GetIndex(  
    [in]  uint64 track_id  
) ;
```

Parameters

- **track_id**

The unique identifier of a track in the playlist.

Return value

If the method succeeds, it returns the one-based index of the track in the playlist. Otherwise, it returns zero, which is an invalid track index.

Remarks

The index of a track in the playlist is always greater than or equal to 1, and less than or equal to the count of tracks in the playlist. The index of a track may be changed by calling [Move\(\)](#) or SetOrder(CINEMO_PLORDER_SHUFFLE) methods of the playlist. The index of a track defines its play order when the play order is set to CINEMO_PLORDER_SEQUENTIAL.

3.18.14 GetTrack

Get track identifier at a given index.

Syntax

```
uint64 GetTrack(  
    [in]  uint32 index  
);
```

Parameters

- **index**

The one-based index of the track in the playlist.

Return value

If the method succeeds, it returns the unique non-zero track ID at the specified index. Otherwise, it returns zero, which is an invalid track ID.

Remarks

The index of a track in the playlist is always greater than or equal to 1, and less than or equal to the count of tracks in the playlist. The index of a track may be changed by calling [Move\(\)](#) or SetOrder(CINEMO_PLORDER_SHUFFLE) methods of the playlist. The index of a track defines its play order when the play order is set to CINEMO_PLORDER_SEQUENTIAL.

3.18.15 GetIndexOfTrackOffset

Get index of track at original (unshuffled) offset within playlist.

Syntax

```
uint32 GetIndexOfTrackOffset(  
    [in]  uint32 offset  
);
```

Parameters

- **offset**

The one-based original offset of the track in the playlist.

Return value

If the method succeeds, it returns the one-based index of the track in the playlist. Otherwise, it returns zero, which is an invalid track index.

Remarks

Both the *index* and *original offset* of a track in the playlist are always greater than or equal to 1, and less than or equal to the count of tracks in the playlist. The *index* is the current position of a track in the playlist, which may change, e.g. if the playlist is shuffled. The *original offset* of a track is the index of the track before any shuffle or other operation occurred, which might cause the index to change. This method is useful when handling *CINEMO_EC_PLAYLIST_CHANGED* events posted from the playlist. This event indicates the start and end offset of contiguous tracks whose metadata has changed. Track indexes cannot be used, since the playlist may be shuffled, in which case indexes are no longer contiguous. Also track identifiers cannot be used, since track identifiers are 64-bit values which are guaranteed to be unique but not necessarily contiguous. The original offset in the playlist is the only way for *CINEMO_EC_PLAYLIST_CHANGED* events to indicate a contiguous block of tracks.

When handling *CINEMO_EC_PLAYLIST_CHANGED* events, an application may determine which tracks have changed by calling [GetIndexOfTrackOffset\(\)](#) for each offset between, and including, the start and end positions indicated in the event.

3.18.16 GetTrackOffsetAtIndex

Get original (unshuffled) offset within playlist of track at index.

Syntax

```
uint32 GetTrackOffsetAtIndex (
    [in]  uint32 index
);
```

Parameters

- **index**

The one-based index of the track in the playlist.

Return value

If the method succeeds, it returns the one-based original offset of the track in the playlist. Otherwise, it returns zero, which is an invalid offset.

Remarks

Both the *index* and *original offset* of a track in the playlist are always greater than or equal to 1, and less than or equal to the count of tracks in the playlist. The *index* is the current position of a track in the playlist, which may change, e.g. if the playlist is shuffled. The *original offset* of a track is the index of the track before any shuffle or other operation occurred, which might cause the index to change.

This method is the inverse of [GetIndexOfTrackOffset\(\)](#).

3.18.17 Open

Open playlist from a URL.

Syntax

```
CinemoError Open(
    [in] const char * szurl,
    [in] uint32 flags,
    [out] CinemoVFSAttributes& attrs
);
```

Parameters

- **szurl**

UTF-8 zero terminated URL of the track.

- **flags**

CINEMO_PLFLAGS bitmask.

- **attrs**

Return attributes of the container.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [Open\(\)](#) method is used to create a new playlist, which consists of all ‘children’ of the specified container URL. The URL could be, for example, a directory pathname, a CD device, or a UPnP browse or search request.

The [Open\(\)](#) method can operate synchronously or asynchronously. To specify synchronous operation, the flags argument should specify *CINEMO_PLFLAGS_WAIT*. Otherwise, the [Open\(\)](#) method will return immediately with *CinemoErrorPending* and the playlist will be opened asynchronously on a background thread. The *CINEMO_EC_PLAYLIST_CHANGED* event can be used to determine when an asynchronous [Open\(\)](#) has completed.

The [Cancel\(\)](#) method can be used to unblock a synchronous [Open\(\)](#) call.

The *szurl* supports the following optional parameters: *index*, *count* and *pagesize*. The *index* and *count* limits the entire range of the playlist. The *pagesize* can be specified for iAP, MTP, AVRCP and UPnP. The page-size is the count of tracks in one browse-operation performed by the playlist. The value is 100 by default.

3.18.18 Close

Close playlist and remove all tracks. Similar function as [RemoveAll\(\)](#) with the ability of asynchronous operation.

Syntax

```
CinemoError Close(  
    [in]  uint32 flags  
) ;
```

Parameters

- **flags**
CINEMO_PLFLAGS bitmask.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If any tracks are currently playing, playback will stop.

The [Close\(\)](#) method can operate synchronously or asynchronously. To specify synchronous operation, the flags argument should specify *CINEMO_PLFLAGS_WAIT*. Otherwise, the [Close\(\)](#) method will return immediately with *CinemoErrorPending* and the playlist will be closed asynchronously on a background thread. The *CINEMO_EC_CLOSE* event can be used to determine when an asynchronous [Close\(\)](#) has completed.

3.18.19 Cancel**3.18.20 Enable**

Cancel or enable synchronous method.

Syntax

```
CinemoError Cancel();  
CinemoError Enable();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [Cancel\(\)](#) method may be called to asynchronously unblock any thread currently blocked inside [Open\(\)](#), [GetURL\(\)](#) or [GetMetapool\(\)](#) methods which were called with the *CINEMO_PLFLAGS_WAIT* flag. In this case, the blocking method will return immediately with *CinemoErrorCancel*.

The [Cancel\(\)](#) method is sticky – after calling [Cancel\(\)](#), the [ICinemoPlaylist](#) object will remain in a cancelled state until [Enable\(\)](#) is called. This is to avoid potential race conditions. The correct sequence of calls to unblock a thread blocked inside e.g. [Open\(\)](#) is as follows:

1. Call [Cancel\(\)](#),
2. Wait for your thread which calls [Open\(\)](#) to unblock,
3. Call [Enable\(\)](#).

3.18.21 AppendTrack

Append a track to the playlist.

Syntax

```
CinemoError AppendTrack(
    [in]  char const * szurl,
    [out] uint64 * ptrack_id = 0
);
```

Parameters

- **szurl**

UTF-8 zero terminated URL of the track.

- **ptrack_id**

Optionally, after appending the track, the assigned track ID is written to ptrack_id.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The track is appended to end of the playlist: that is, after calling this method, the index of the track in the playlist will be equal to the count of tracks in the playlist. If the method succeeds, the track will be assigned a unique track identifier. The calling application should make no assumptions about the value of the track identifier, except that it is guaranteed to be unique amongst all tracks in the playlist. To obtain the track identifier, the following code sequence can be used: Note that this method should not be used with CD audio discs. To create a playlist containing all CD audio tracks from a disc, call [ICinemoPlaylist::Open\(\)](#) with the disc path.

```
/* append a track */

CinemoError hr;

if (hr = pplaylist->AppendTrack(szurl))
{
    return hr; /* return error code */
}

/* track_id from index */

uint32 track_id = pplaylist->GetTrack(pplaylist->GetCount());

/* do something with the track_id */

....
```

3.18.22 AppendTrackWithID

Append a track to the playlist with a user defined unique ID.

Syntax

```
CinemoError AppendTrackWithID (
    [in]  uint64 track_id,
    [in]  char const * szurl
);
```

Parameters

- **track_id**
User-defined 64-bit identifier of the track to append.
- **szurl**
UTF-8 zero terminated URL of the track.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Note that zero is not a valid track ID and cannot be used. If a track is attempted to be added with a track ID of zero, the method will fail with *CinemoErrorArguments*.

Note that this method cannot be used with CD Audio discs, which are treated by Cinemo as folders that contain tracks. The track names that can be used to add CD Audio disc tracks to a playlist can be obtained using the [ICinemoMetapool](#) interface. The track is appended to end of the playlist: that is, after calling this method, the index of the track in the playlist will be equal to the count of tracks in the playlist. The track identifier must be unique amongst all other tracks in the playlist. If a track with the same identifier is already present, the method will fail with *CinemoErrorArguments*.

3.18.23 AppendSource

Append a source object to the playlist.

Syntax

```
CinemoError AppendSource(  
    [in] struct ICinemoUnknown * psource  
) ;
```

Parameters

- **psource**

Pointer to a user-provided source object.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The method adds a user-provided source object to the playlist. During playback initialization the source object will be queried for a range of supported interfaces in order to determine the appropriate way of retrieving data from it. Currently the object must implement either of the following interfaces:

- **ICinemoFile**

A custom-implemented object that derives from [ICinemoFile](#). In order to function properly the object implementation must fully support all [ICinemoUnknown](#) methods. An example is provided with the sample application [SamplePlayer_ICinemoFile](#).

- **ICinemoLiveStream**

A [CinemoLiveStream](#) object that was previously created by [CinemoCreateLiveStream\(\)](#)

- **ICinemoProjection**

A [CinemoProjection](#) object that was previously created by [CinemoCreateProjection\(\)](#)

The track is appended to end of the playlist: that is, after calling this method, the index of the track in the playlist will be equal to the count of tracks in the playlist.

3.18.24 Remove

Remove tracks with given identifiers from the playlist.

Syntax

```
CinemoError Remove (
    [in] const uint64 * ptracks,
    [in] uint32 ntracks
);
```

Parameters

- **ptracks**

Pointer to an array of unique track IDs.

- **ntracks**

The number of tracks in the array.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method loops through the given array of track identifiers and removes each one from the playlist. Invalid track identifiers will be ignored. When tracks are removed from the playlist, the remaining tracks are re-indexed, such that the index of each remaining track is greater than or equal to 1, and less than or equal to the number of tracks.

Tracks may be removed from the playlist even if they are currently playing. In this case, playback of the track will stop; the subsequent playback behavior depends on the current assigned values of *CINEMO_PLORDER* and *CINEMO_PLREPEAT*, as if the track had ended naturally.

3.18.25 RemoveAll

Remove all tracks.

Syntax

```
CinemoError RemoveAll();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If any tracks are currently playing, playback will stop.

3.18.26 Move

Move tracks to a specified index.

Syntax

```
CinemoError Move (
    [in]  uint32 index,
    [in]  const uint64 * ptracks,
    [in]  uint32 ntracks
);
```

Parameters

- **index**

The one-based target index in the playlist.

- **ptracks**

Pointer to an array of unique track ID's.

- **ntracks**

The number of tracks in the array.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.18.27 Find

Get track identifier of a given URL if it exists in the playlist.

Syntax

```
uint64 Find(  
    [in] const char * szurl  
);
```

Parameters

- **szurl**
UTF-8 zero-terminated URL of the track.

Return value

If the method succeeds, it returns the unique non-zero track ID with the specified URL. Otherwise, it returns zero, which is an invalid track ID.

Remarks

This method may be used to determine if a track already exists in the playlist.

3.18.28 GetMetapool

Get metapool for a track.

Syntax

```
CinemoError GetMetapool(
    [in]  uint64 track_id,
    [in]  uint32 flags,
    [out] struct ICinemoMetapool ** pp
);
```

Parameters

- **track_id**

The unique identifier of a track in the playlist.

- **flags**

CINEMO_PLFLAGS bitmask.

- **pp**

The address of a pointer to receive the [ICinemoMetapool](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

For manually created playlists, i.e. constructed with [AppendTrack\(\)](#) or [AppendTrackWithID\(\)](#) calls, [GetMetapool\(\)](#) will return, for each track, an initially empty [ICinemoMetapool](#) instance which may be written to by the calling application. The returned [ICinemoMetapool](#) is persistent for each track, which means the same [ICinemoMetapool](#) pointer will be returned if [GetMetapool\(\)](#) is called more than once with the same *track_id*.

This method allows the calling application to provide additional metadata about a track, by writing to the returned [ICinemoMetapool](#) interface (see [ICinemoMetapool::AddItem\(\)](#) methods). For example, the metadata items *CINEMO_METANAME_VFS_NAME* and *CINEMO_METANAME_VFS_ICON* may be used by attached [ICinemoPlayer](#) instances to provide additional information to the distributed playback discovery service. The metadata is read by attached [ICinemoPlayer](#) instances whenever playback of a new track is started, or [ICinemoPlayer::SetDistributed\(\)](#) is called.

For playlists created with [Open\(\)](#), the [GetMetapool\(\)](#) method will return an [ICinemoMetapool](#) instance initialized with all known metadata for the specified track. For example, if the playlist is opened with a CD device path, the metadata for each track may be automatically initialized with CD_TEXT contents. Depending on the type of container opened, the metadata may not be known at the time of calling this method. For example, UPnP playlists may contain thousands of tracks, and for performance reasons the metadata for each track is not fetched from the UPnP media server until it is needed.

The *flags* argument specifies what to do if the metadata is not known:

- **CINEMO_PLFLAGS_NONE**

Do nothing, return *CinemoErrorPending* error code.

- **CINEMO_PLFLAGS_READ**

Trigger an asynchronous fetch of the metadata (e.g. from the UPnP server), and return immediately with *CinemoErrorPending* error code. *CINEMO_EC_PLAYLIST_CHANGED* event will be posted when the URL is successfully fetched.

- **CINEMO_PLFLAGS_READ | CINEMO_PLFLAGS_WAIT**

Trigger a synchronous fetch of the metadata from the server, and return *CinemoNoError* when the metadata is successfully fetched. This may be unblocked by [Cancel\(\)](#), in which case the method will return *CinemoErrorCancel*.

3.18.29 GetURL

Get URL of a track.

Syntax

```
CinemoError GetURL(
    [in]  uint64 track_id,
    [in]  uint32 flags,
    [out] struct ICinemoUTF8 ** pp
);
```

Parameters

- **track_id**

The unique identifier of a track in the playlist.

- **flags**

CINEMO_PLFLAGS bitmask.

- **pp**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

For manually created playlists, i.e. constructed with [AppendTrack\(\)](#) or [AppendTrackWithID\(\)](#) calls, [GetURL\(\)](#) returns the same URL specified when the track was added to the playlist.

For playlists created with [Open\(\)](#), the [GetURL\(\)](#) method will return the absolute URL of a track. This is a potentially blocking call, which can be unblocked with [Cancel\(\)](#).

Depending on the type of container opened, the URL may not be known at the time of calling this method. For example, UPnP playlists may contain thousands of tracks, and for performance reasons the URL of each track is not fetched from the UPnP media server until needed.

The *flags* argument specifies what to do if the URL is not known:

- **CINEMO_PLFLAGS_NONE**

Do nothing, return *CinemoErrorPending* error code.

- **CINEMO_PLFLAGS_READ**

Trigger an asynchronous fetch of the URL (e.g. from the UPnP server), and return immediately with *CinemoErrorPending* error code. *CINEMO_EC_PLAYLIST_CHANGED* event will be posted when the URL is successfully fetched.

- **CINEMO_PLFLAGS_READ | CINEMO_PLFLAGS_WAIT**

Trigger a synchronous fetch of the URL from the server, and return *CinemoNoError* when the URL

is successfully fetched. This may be unblocked by [Cancel\(\)](#), in which case the method will return *CinemoErrorCancel*.

HUMAX Confidential

3.18.30 Select

Select the entire playlist for playback on the underlying device.

Syntax

```
CinemoError Select(
    [in] const CinemoSelectParams& params,
    [in] uint32 flags
);
```

Parameters

- **params**

`CinemoSelectParams` to specify additional parameters.

- **flags**

`CINEMO_PLFLAGS` bitmask.

Return value

If the method succeeds, it returns `CinemoNoError`. Otherwise, it returns a `CinemoError` code.

Remarks

When browsing local Apple or Bluetooth devices using the `ICinemoPlaylist` interface, their metapools will be flagged as `CINEMO_VFS_FLAG_SELECTABLE`. This indicates, that [Select\(\)](#) may be called to play the contents of the playlist on the underlying device.

When browsing UPnP servers, the [Select\(\)](#) call may also be used to send the contents of playlist to an underlying Apple device. However, there will be no `CINEMO_VFS_FLAG_SELECTABLE` flag available, indicating the availability of this method. In order to perform the [Select\(\)](#) command properly, the path of each track needs to fulfill the following criteria:

- It consists of a valid path for the underlying device with either "iap://" or "iaptrack://" protocol, e.g.
"iaptrack:///dev/bus/usb/003/005". (DDP is also supported)
- and the Apple UID of the track and the media library identifier are appended as URL parameters, e.g.
"iaptrack:///dev/bus/usb/003/005?uid=2171750826250074286&lib=7F48D605-90B7-4B14-A006-9E7F2522C22A-8.1.2".
- or the persistent identifier of a collection on the Apple device, the media library identifier and the collection type are appended as URL parameters, e.g.
"iap:///dev/bus/usb/003/005?cid=13062362167465469298&lib=7F48D605-90B7-4B14-A006-9E7F2522C22A-8.1.2&ctype=4".

Optionally a collection start index can be appended if the collection type is playlist, which initiates playback from the indexed track in the playlist. The collection type (ctype) values are 0: playlist, 1: artist, 2: album, 3: album artist, 4: genre, 5: composer.

To play all tracks available on an iAP2 device the special collection type 6 (which does not support a

starting index) may be used:

```
"iap:///dev/bus/usb/003/005?cid=1&ctype=6&lib=7F48D605-90B7-4B14-A006-9E7F2522C22A-8.1.2"
```

Tracks not corresponding to the criteria above in their path will be ignored. The device path of the first item in the playlist will be used to open the underlying device. All tracks with different paths are also ignored. In case a collection shall be played, the playlist shall contain only one track with the cid parameter. If more than one track is encountered, then an error is reported.

Calling [Select\(\)](#) will cancel any further updates on the playlist from UPnP queries while it Cinemo is gathering metadata.

This method is functionally equivalent to [ICinemoTrackSelector::Select\(\)](#) method on the playlist's underlying [ICinemoVFS](#). Please refer to [ICinemoTrackSelector](#) for further information.

The [Select\(\)](#) method can operate synchronously or asynchronously. To specify synchronous operation, the flags argument should specify *CINEMO_PLFLAGS_WAIT*. Otherwise, the [Select\(\)](#) method will return immediately with *CinemoErrorPending* and the selection will be opened asynchronously on a background thread. The *CINEMO_EC_SELECT* event can be used to determine when an asynchronous [Select\(\)](#) has completed.

Depending on the iOS version and the type of tracks selected, not all of them may show up in the “Now Playing” playlist of the Music App on the Apple device. Especially podcasts and iTunes U episodes may not be displayed in certain iOS versions. Playback of these items, however, is working as intended.

For iAP2 devices there is a maximum number of tracks which can be selected from a single playlist. This number depends on the maximum payload size of an iAP2 packet which the device can receive. This payload size is available in the metapool of the device with the metaname *CINEMO_METANAME_IAP_OUTGOING_MAX_PAYLOAD_SIZE*. Each track id in a Select() call occupies 8 bytes, so roughly 8100 songs can be selected at once for the maximum allowed payload size of 65525 bytes. If the number of tracks is too large for a single packet this method will return with *CinemoErrorOverflow*.

Please also note that not all Apple iAP1 devices support selecting custom playlists (e.g., created by browsing Cinemo Media Management). Selecting tracks/folders from the native device browse hierarchy is always available.

It is possible to determine if an Apple iAP1 device supports custom playlist selection by looking at the metadata of such a device:

```
# SampleMetadata "iap://usb:///dev/bus/usb/001/007"
opening...

path: iap://usb:///dev/bus/usb/001/007
type: FOLDER | VIRTUAL | ORDERED | WATCHABLE | SELECTABLE | USB | IAP | iAP-1
title 00 index 00 VFS_UUID: "76bebbad1c26ea7c5a70c6e075bee2b03bb7610e"
title 00 index 00 VFS_NAME: "cinemo@s iPhone"
title 00 index 00 VFS_USB_VENDOR_ID: 1452
title 00 index 00 VFS_USB_PRODUCT_ID: 4776
title 00 index 00 VFS_USB_MANUFACTURER: "Apple Inc."
title 00 index 00 VFS_USB_PRODUCT: "iPhone"
```

```
title 00 index 00 VFS_USB_SERIAL_NO: "76bebbad1c26ea7c5a70c6e075bee2b03bb7610e"  
title 00 index 00 VFS_IAP1_IPOD_VERSION: "8.1.2"  
title 00 index 00 VFS_IAP1_LINGO_OPTIONS: 26808939519  
title 00 index 03 VFS_IAP1_LINGO_OPTIONS: 5  
title 00 index 04 VFS_IAP1_LINGO_OPTIONS: 238  
title 00 index 10 VFS_IAP1_LINGO_OPTIONS: 4  
...
```

All capabilities of a device will be listed as a bitmask in the VFS_IAP1_LINGO_OPTIONS metanames. The item with index 4 (CINEMO_IAP1_LINGO_EXTENDED) represents the “Extended Lingo”, which is used for UID-based command support. If the seventh bit of this bitmask is set (0x40), the device supports UID-based commands and folder/track selection is supported. Otherwise, calling [Select\(\)](#) (or [ICinemoPlaylist::SelectTrack\(\)](#)) for custom playlists on this device will fail with *CinemoErrorUnexpected* (Note: the values in the printout above are in decimal).

For certain Apple devices the VFS_IAP1_LINGO_OPTIONS metaname may not be available. In this case the VFS_IAP1_LINGO_VERSION metaname can be used to determine if the device supports custom playlist selection. If the lingo version with index 4 (CINEMO_IAP1_LINGO_EXTENDED) is greater or equal to 269, the device supports custom playlist selection. All devices with a lower version do not support custom playlist selection.

According to Apple specifications, it is undefined behavior to call [ICinemoPlaylist::Select\(\)](#) or [ICinemoPlaylist::SelectTrack\(\)](#) on a playlist containing more than one type of track. In particular, these types are music tracks, podcasts, audiobooks and iTunesU episodes. Cinemo will determine the type of track that has been selected and remove all other types from the selection. For [ICinemoPlaylist::Select\(\)](#), the first track is used as the type of track to select, while for [ICinemoPlaylist::SelectTrack\(\)](#) the selected track is used as reference. To avoid mixing types of tracks, the application may choose to avoid displaying lists that contain more than one type of track. This could be done with search queries or custom browse hierarchies with Cinemo Media Management, for example.

3.18.31 SelectTrack

Select a single track for playback on the underlying device.

Syntax

```
CinemoError SelectTrack(
    [in]  uint64 track_id,
    [in]  const CinemoSelectParams& params,
    [in]  uint32 flags
);
```

Parameters

- **track_id**

The 64-bit identifier of the track to select.

- **params**

`CinemoSelectParams` to specify additional parameters.

- **flags**

`CINEMO_PLFLAGS` bitmask.

Return value

If the method succeeds, it returns `CinemoNoError`. Otherwise, it returns a `CinemoError` code.

Remarks

When browsing local Apple or Bluetooth devices using the `ICinemoPlaylist` interface, their metapools will be flagged as `CINEMO_VFS_FLAG_SELECTABLE`. This indicates, that [SelectTrack\(\)](#) may be called to play the contents of the playlist on the underlying device, starting at index `track_id`.

When browsing UPnP servers, the [SelectTrack\(\)](#) call may also be used to send the contents of playlist to an underlying Apple device. However, there will be no `CINEMO_VFS_FLAG_SELECTABLE` flag available, indicating the availability of this method. In order to perform the [SelectTrack\(\)](#) command properly, the path of each track needs to fulfill the following two criteria:

- It consists of a valid path for the underlying device with either “`iap://`” or “`iaptrack://`” protocol, e.g. “`iaptrack:///dev/bus/usb/003/005`”. (DDP is also supported)
- The Apple UID of the track and the media library identifier are appended as URL parameters, e.g. “`iaptrack:///dev/bus/usb/003/005?uid=2171750826250074286&lib=7F48D605-90B7-4B14-A006-9E7F2522C22A-8.1.2`”.

Tracks not corresponding to the criteria above in their path will be ignored. The device path of the first item in the playlist will be used to open the underlying device. All tracks with different paths are also ignored.

Calling [SelectTrack\(\)](#) will cancel any further updates on the playlist from UPnP queries.

This method is functionally equivalent to [ICinemoTrackSelector::SelectChild\(\)](#) method on the playlist's underlying [ICinemoVFS](#). Please refer to [ICinemoTrackSelector](#) for further information.

The [SelectTrack\(\)](#) method can operate synchronously or asynchronously. To specify synchronous operation, the flags argument should specify `CINEMO_PLFLAGS_WAIT`. Otherwise, the [SelectTrack\(\)](#) method will return immediately with `CinemoErrorPending` and the selection will be opened asynchronously on a background thread. The `CINEMO_EC_SELECT` event can be used to determine when an asynchronous [SelectTrack\(\)](#) has completed.

Please see the [ICinemoPlaylist::Select\(\)](#) method for Apple limitations on [SelectTrack\(\)](#).

When [SelectTrack\(\)](#) is called on the "NowPlaying" playlist of an Apple device it works like [ICinemoPlayer2::SeekTitleChapter\(\)](#). Only the index will be sent to the device and not the entire list. That is also why `CinemoSelectParams` are not supported for the "NowPlaying" playlist.

3.18.32 InitPlayback

Get [ICinemoPlayer](#) interface for playback.

Syntax

```
CinemoError InitPlayback(  
    [out] struct ICinemoPlayer ** pp  
) ;
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoPlayer](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method returns a reference counted pointer to an [ICinemoPlayer](#) interface capable of playing any track in the playlist. To play a track in the playlist, check the [ICinemoPlayer](#) interface documentation. The interface should be released when it is no longer required.

Note there is no imposed limit to the number of [ICinemoPlayer](#) interfaces. It is allowed to obtain multiple interfaces and simultaneously play multiple tracks either from the same playlist, or from different playlists. There may however be practical reasons not to do this, depending on the available CPU, memory, video and audio resources of the target platform.

Note also, the created [ICinemoPlayer](#) interface holds its own reference count on the [ICinemoPlaylist](#) interface which created it. The calling application is therefore free to release the [ICinemoPlaylist](#) interface, and as long as the [ICinemoPlayer](#) interface still exists, it will be able to continue playing tracks from the playlist.

3.18.33 InitPlayback2

Get [ICinemoPlayer2](#) interface for playback.

Syntax

```
CinemoError InitPlayback2(
    [out] struct ICinemoPlayer2 ** pp
);
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoPlayer2](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method returns a reference counted pointer to an [ICinemoPlayer2](#) interface capable of playing any track in the playlist. To play a track in the playlist, check the [ICinemoPlayer2](#) interface documentation. The interface should be released when it is no longer required.

Note there is no imposed limit to the number of [ICinemoPlayer2](#) interfaces. It is allowed to obtain multiple interfaces and simultaneously play multiple tracks either from the same playlist, or from different playlists. There may however be practical reasons not to do this, depending on the available CPU, memory, video and audio resources of the target platform.

Note also, the created [ICinemoPlayer2](#) interface holds its own reference count on the [ICinemoPlaylist](#) interface which created it. The calling application is therefore free to release the [ICinemoPlaylist](#) interface, and as long as the [ICinemoPlayer2](#) interface still exists, it will be able to continue playing tracks from the playlist.

3.18.34 GetAlphaIndexCount

Get count of available alphabetical index items.

Syntax

```
uint32 GetAlphaIndexCount();
```

Return value

Return the count of alphabetical index items.

Remarks

Alphabetical indexes are currently only available when opening UPnP playlists supplied by Cinemo Media Management server. The index is provided by the server.

The count of alphabetical index items may change when a playlist automatically updates. Calling applications should monitor *CINEMO_EC_PLAYLIST_CHANGED* events posted by the playlist, and update the user interface if any change is detected.

3.18.35 GetAlphaIndex

Get alphabetical index item.

Syntax

```
CinemoError GetAlphaIndex(
    [in]      uint32 index,
    [in/out] CinemoAlphaIndex& attrs
);
```

Parameters

- **index**

The one-based index of the alphabetical index item.

- **attrs**

The alphabetical index item.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The specified index is an index into the list of alphabetical index items (not the list of tracks). It must be greater or equal to 1, and less or equal to the number of alphabetical index items returned by [GetAlphaIndexCount\(\)](#).

The *CinemoAlphaIndex* structure is defined here:

```
typedef struct {
    struct ICinemoUTF8 * pname;      /* symbol */
    uint32 index;                  /* index within playlist */
} CinemoAlphaIndex;
```

Ensure to set the *pname* field to a valid [ICinemoUTF8](#) interface pointer **before** calling this method. The alphabetical index symbol will be copied to the supplied interface as a UTF-8 string. The string is guaranteed to be a single upper-case character which may consist of multiple bytes (e.g. Japanese). The calling application is responsible for allocating and releasing the [ICinemoUTF8](#) interface (see *CinemoCreateUTF8* function).

The returned *index* field is the one-based index into the playlist of the first track whose title starts with the given symbol. This index is always greater than or equal to 1, and less than or equal to the number of tracks in the playlist.

3.18.36 GetCandidateCount

Get count of available next character candidates when performing text queries.

Syntax

```
uint32 GetCandidateCount();
```

Return value

Return the count of next character candidates.

Remarks

Candidates are currently only available when opening UPnP playlists supplied by Cinemo Media Management server. The candidates are provided by the server.

Candidates are only available if the URL parameter *candidates=1* is specified in the query URL (see 2.2.3).

The count of candidates may change when a playlist automatically updates. Calling applications should monitor *CINEMO_EC_PLAYLIST_CHANGED* events posted by the playlist, and update the user interface if any change is detected.

3.18.37 GetCandidate

Get next character candidate for text queries.

Syntax

```
CinemoError GetCandidate(
    [in]      uint32 index,
    [in/out] CinemoQueryCandidate& attrs
);
```

Parameters

- **index**

The one-based index of the candidate.

- **attrs**

The candidate.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The specified index is an index into the list of next character candidates (not the list of tracks). It must be greater or equal to 1, and less or equal to the number of candidates returned by [GetCandidateCount\(\)](#). The *CinemoQueryCandidate* structure is defined here:

```
typedef struct {
    struct ICinemoUTF8 * pname;          /* candidate symbol */
    uint32 prio;                      /* priority of candidate */
} CinemoQueryCandidate;
```

Ensure to set the *pname* field to a valid [ICinemoUTF8](#) interface pointer **before** calling this method. The candidate symbol will be copied to the supplied interface as a UTF-8 string. The calling application is responsible for allocating and releasing the [ICinemoUTF8](#) interface (see *CinemoCreateUTF8* function).

The returned *prio* field is the priority of the candidate calculated from the frequency of the character.

3.18.38 Freeze

Set freeze state of the playlist.

Syntax

```
CinemoError Freeze(  
    [in]  CINEMO_PLFREEZE freeze  
) ;
```

Parameters

- **freeze**

The new freeze state

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The freeze state defines the behaviour of a watchable playlist. It can be set at any time, even before opening the playlist. The freeze state is permanent as long as the playlist instance exists. It may be set to one of the following values:

- **CINEMO_PLFREEZE_NONE**

Not frozen, dynamic updates are allowed.

- **CINEMO_PLFREEZE_TRACKS**

Playlist tracks are frozen, but dynamic metadata update for the tracks are allowed.

- **CINEMO_PLFREEZE_TRACKS_AND_METADATA**

Playlist tracks and metadata are frozen

After the *Freeze()* call, the new freeze state is active, e.g. if the new state is *CINEMO_PLFREEZE_TRACKS* there will be no track changes. Then after changing the freeze state to *CINEMO_PLFREEZE_NONE* all changes that occurred while the Playlist was frozen are applied.

3.18.39 AttachPlayer

Attach an existing [ICinemoPlayer](#) to the playlist and detach that [ICinemoPlayer](#) from any playlist it was attached to before.

Syntax

```
CinemoError AttachPlayer(  
    [in]    ICinemoPlayer* pplayer,  
    [in]    uint32 reserved  
);
```

Parameters

- **pplayer**
The [ICinemoPlayer](#) interface to attach to the playlist
- **reserved**
reserved, must be 0

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the [ICinemoPlayer](#) is currently playing a track, then this track is closed and the playback stops, unless a track with the same ID exists in this [ICinemoPlaylist](#). That track is assumed to be the same track, which is true for playlists created by browse or search requests on the same MM server.

3.18.40 AttachPlayer2

Attach an existing [ICinemoPlayer2](#) to the playlist and detach that [ICinemoPlayer2](#) from any playlist it was attached to before.

Syntax

```
CinemoError AttachPlayer2(  
    [in]    ICinemoPlayer2* pplayer2,  
    [in]    uint32 reserved  
);
```

Parameters

- **pplayer2**
The [ICinemoPlayer2](#) interface to attach to the playlist
- **reserved**
reserved, must be 0

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the [ICinemoPlayer2](#) is currently playing a track, then this track is closed and the playback stops, unless a track with the same ID exists in this [ICinemoPlaylist](#). That track is assumed to be the same track, which is true for playlists created by browse or search requests on the same MM server.

3.19 ICinemoPlayer

The [ICinemoPlayer](#) interface is used to play tracks.

Note: There is now an alternative interface [ICinemoPlayer2](#) which supports playback of all files and media types supported by this [ICinemoPlayer](#) interface. The [ICinemoPlayer](#) interface is not sufficient to obtain a certifiable playback of blu-ray discs or iAP devices. When used with blu-ray the methods of the [ICinemoPlayer](#) interface change to the behavior of the corresponding method in the [ICinemoPlayer2](#) interface. When used with iAP devices, the methods will become fully asynchronous.

This thread-safe interface may be obtained in two ways:

- **Playlist mode**

Call [ICinemoPlaylist::InitPlayback\(\)](#) on an instantiated playlist. First it is necessary to create an [ICinemoPlaylist](#). There is no limit to the number of players which may be obtained from a single playlist. Each player maintains its independent playback state, so it is possible to play multiple tracks from the same playlist at the same time.

- **Standalone mode**

Call [CinemoCreatePlayer\(\)](#) with an [ICinemoTrackProvider](#) interface provided by the calling application. In this case, the application is responsible for maintaining tracks. In all other respects, the operation of the player is the same.

ICinemoPlayer Methods

METHOD	DESCRIPTION
SetEventCallback	Set callback function to receive player events
SetEventQueue	Set event queue to receive player events
SetEventSourceID	Set source ID for posted events
SetGraphParams	Set graph creation parameters
SetAudioParams	Set audio device parameters
GetAudioParams	Get audio device parameters
SetVideoParams	Set video device parameters
GetVideoParams	Get video device parameters
SetDistributed	Set distributed playback parameters
SetSpeed	Set playback speed
SetScanTime	Set the scan time
SetTimeEventsInterval	Set the interval between periodic <i>CINEMO_EC_TIME</i> events
SetAudioCaptureCallback	Set callback function to receive PCM audio samples
OpenTrack	Open a track, but do not start playback
CloseTrack	Close the current track
PlayTrack	Open a track, and automatically start playback
NextTrack	Skip to the next track

METHOD	DESCRIPTION
<u>PreviousTrack</u>	Skip to the previous track
<u>Play</u>	Play the current track (if in stopped state)
<u>Stop</u>	Stop the current track
<u>Seek</u>	Seek to a time position within current track
<u>SeekTitleChapter</u>	Seek to a title or chapter
<u>ShowMenu</u>	Show menu
<u>ResumeTitle</u>	Resume title playback (after Stop or ShowMenu)
<u>ReturnFromSubmenu</u>	Navigate up through menu hierarchy
<u>NextChapter</u>	Skip to the next chapter
<u>PreviousChapter</u>	Skip to the previous chapter
<u>ReplayChapter</u>	Replay current chapter from beginning
<u>SetRepeatChapter</u>	Set repeat mode of the chapter
<u>SetRepeatTitle</u>	Set repeat mode of the title
<u>StepForward</u>	Step forward one frame
<u>StepBackward</u>	Step backward one frame
<u>SelectAudio</u>	Select an audio stream
<u>SelectAngle</u>	Select an angle or video stream
<u>SelectSubpicture</u>	Select a subpicture stream
<u>AddExternalSubtitle</u>	Add subtitles loaded from an external resource
<u>SelectButton</u>	Select a button
<u>ActionButton</u>	Action currently selected button
<u>SelectButtonRelative</u>	Select button relative to the currently selected button
<u>SelectButtonPosition</u>	Select button at XY position
<u>ActionButtonPosition</u>	Action button at XY position
<u>AcceptParentalLevel</u>	Reply to currently pending temporary parental level change request
<u>AcceptCMI</u>	Reply to currently pending content management information (CMI) change request
<u>GetStatus</u>	Get current playback status
<u>GetGraphStatus</u>	Get current decoder graph status
<u>GetQualityInfo</u>	Get current audio and video renderer quality info
<u>GetPosition</u>	Get play position within current track
<u>GetDuration</u>	Get duration of current track
<u>GetBuffered</u>	Get buffer status of the current track
<u>GetAudio</u>	Get audio stream attributes
<u>GetAngle</u>	Get angle or video stream attributes

METHOD	DESCRIPTION
<u>GetSubpicture</u>	Get subpicture attributes
<u>GetMediaInfo</u>	Get information about the medium
<u>Title</u>	Get information about a title
<u>GetChapter</u>	Get information about a chapter
<u>SetSessionData</u>	Set session data for the distributed playback server or client
<u>GetSessionPool</u>	Get the distributed playback session pool
<u>InitMetapool</u>	Obtain an <u>ICinemoMetapool</u> interface for reading metadata
<u>InitPlaylistMetapool</u>	Obtain an <u>ICinemoMetapool</u> interface for reading the playlist metadata for the current track
<u>InitPainter</u>	Obtain an <u>ICinemoPainter</u> interface for painting video
<u>SaveState</u>	Save current playback state
<u>RestoreState</u>	Restore current playback state
<u>SignalPlaylistChanged</u>	Signal that the parent playlist has changed
<u>TakeSnapshot</u>	Take a snapshot of the current video frame
<u>GetWindowDeviceName</u>	Get window device name string with platform specific window or surface handles as URL parameters.
<u>GetTrackURL</u>	Get URL for the currently playing track from the playlist or track provider.
<u>GetTrackVFSAttributes</u>	Get VFS attributes for the currently playing track.
<u>GetDistributedServerURL</u>	Get the URL of the Cinemo Distributed Playback Server

3.19.1 SetEventCallback

Set callback function to receive player events.

Syntax

```
CinemoError SetEventCallback(  
    [in]    void * puser,  
    [in]    CinemoEventCallback pcall  
) ;
```

Parameters

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The user defined application callback function will be called asynchronously whenever a new player event is available. The application is responsible for synchronizing events from multiple threads to its own application thread ([ICinemoEventQueue](#) interface may be used for this purpose).

To stop receiving events, a NULL callback function may be specified.

3.19.2 SetEventQueue

Set event queue to receive player events.

Syntax

```
CinemoError SetEventQueue(  
    [in]     struct ICinemoEventQueue * pqueue  
) ;
```

Parameters

- **pqueue**

pointer to [ICinemoEventQueue](#) object to receive player events.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This approach helps transferring the events to the application main thread using an event queue object. It behaves the same as if an application callback would push the new events to the event queue. The queue object will be kept referenced by the player until it is set again or the player is destroyed.

To stop associating a queue with the player, a NULL pointer may be specified.

3.19.3 SetEventSourceID

Set source ID for posted events.

Syntax

```
CinemoError SetEventSourceID(  
    [in]      uint32 source_id  
) ;
```

Parameters

- **source_id**

User-defined number passed in *CinemoEvent* *source_id* member.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This approach helps differentiate sources of events when using shared event queue object.

3.19.4 SetGraphParams

Set graph creation parameters.

Syntax

```
CinemoError SetGraphParams(  
    [in]      const CinemoGraphParams& params  
) ;
```

Parameters

- **params**

Graph creation parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The graph parameters should be set before starting playback.

3.19.5 SetAudioParams

Set audio device parameters.

Syntax

```
CinemoError SetAudioParams(
    [in]      const CinemoAudioParams& params
);
```

Parameters

- **params**

Audio device parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This synchronous method changes the parameters for the audio output device plugin. The audio device plugin is configured separately by the *CINEMO_OPTION_PLUGIN_AUDIO_DEVICE* configuration option. The *CinemoAudioParams* is defined by the following structure:

```
typedef struct {
    char device_name[256];
    uint32 device_number;
    uint32 disabled;
    uint32 crc32;
    CinemoAudioProperties props;
} CinemoAudioParams;
```

- **device_name**

This field is passed as a string parameter to the audio device plugin. By default, *device_name* is an empty string. On Windows, *device_name* specify the name of the DirectSound device to use for audio output.

- **device_number**

Support for cloned secondary audio device feature. It should be only used after consultation with Cinemo. By default, zero for primary device parameters.

- **disabled**

Disables the audio output device for this [ICinemoPlayer](#) instance. Playback will continue, but no audio will be output. Unlike calling *SelectAudio(0)*, setting the *disabled* field will detach the audio output device from the [ICinemoPlayer](#) instance. The resources for the audio output device are freed unless being used by another [ICinemoPlayer](#) or other Cinemo SDK interface instance.

- **crc32**

Indicates the audio device should generate CRC32 quality info.

The CinemoAudioProperties structure is defined here:

```
typedef struct {
    float volume_level;
    uint32 volume_fade_ms;
    sint32 balance;
    sint32 fade;
    uint32 mute;
    uint32 channelclone;
    CinemoAudioPropertiesFlags flags;
    char device_params[256];
} CinemoAudioProperties;
```

```
typedef struct {
    uchar channelclone_dualmono_only : 1;
    uchar reserved[3];
} CinemoAudioPropertiesFlags;
```

- **volume_level**

Specifies the audio volume level in a range from 0 (muted) to 1 (full range). The default volume level is 1.

- **volume_fade_ms**

Defines the time for fading from the current volume to the specified target volume in milliseconds (the allowed range is 0-3000 ms). It is **strongly recommended to set reasonable (e.g. 10 ms) value** in order to prevent artefacts on volume change. The configuration property is only used if the option *CINEMO_OPTION_AUDIO_DEVICE_ENABLE_HW_VOLUME* is set to FALSE.

- **balance**

Specifies the audio left-right balance. The allowed range is from -50 (only left channel(s) audible) to +50 (only right channel(s) audible). Using the default value 0 will result in unchanged left-right balance, as compared to the source.

- **fade**

Specifies the audio front-rear balance. The allowed range is from -50 (only front channel(s) audible) to +50 (only rear channel(s) audible). Using the default value 0 will result in unchanged front-rear balance, as compared to the source.

- **mute**

Specifying a non-zero value will result in the audio being muted. The **volume_fade_ms** is applied.

- **channelclone**

Specifies channel number to clone. The allowed range is from 0 (channel cloning disabled) to the number of output channels. If channelclone is non-zero, the PCM signal from the specified channel number will be duplicated to all other audio channels. The channel numbers are counted in order of their bitmask in *CINEMO_CHANNELCONFIG_xx*.

- **flags.channelclone_dualmono_only**

Specifies if channel clone is active for the streams not identified as dual mono.

- **device_params**

Is used to pass runtime specific parameters to the audio device.

3.19.6 GetAudioParams

Get the current audio device parameters.

Syntax

```
CinemoError GetAudioParams(  
    [out] CinemoAudioParams& params  
) ;
```

Parameters

- **params**

Audio device parameter structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.19.7 SetVideoParams

Set video device parameters.

Syntax

```
CinemoError SetVideoParams(
    [in]      const CinemoVideoParams& params
);
```

Parameters

- **params**

Video device parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This synchronous method changes the parameters for the video device plugin. The video device plugin is configured separately by the *CINEMO_OPTION_PLUGIN_VIDEO_DEVICE* configuration option. If there are multiple [ICinemoPlayer](#) instances with the same video device_name, only one video device plugin instance will be created, and the video output from the different [ICinemoPlayer](#) instances will be dynamically mixed to the one video device. The *CinemoVideoParams* is defined here:

```
typedef struct {
    char device_name[256];
    uint32 device_number;
    uint32 disabled;
    uint32 crc32;
    CinemoVideoProperties props;
} CinemoVideoParams;
```

- **device_name**

This field is passed as a string parameter to the video device plugin. By default, *device_name* is an empty string.

- **device_number**

Support for cloned secondary video device feature. It should be only used after consultation with Cinemo. By default, zero for primary device parameters.

- **disabled**

Disables the video output device for this [ICinemoPlayer](#) instance. Playback will continue, but no video will be output. Setting the disabled flag will detach the video device from the [ICinemoPlayer](#) instance. The resources for the video output device are released unless being used by another [ICinemoPlayer](#) or other Cinemo SDK interface instance.

- **crc32**

Indicates the video device should generate CRC32 quality info.

The CinemoVideoProperties structure is defined here:

```
typedef struct {
    sint32 zorder;
    sint32 reserved[4];
    CINEMO_ASPECTMODE aspect_mode;
    CINEMO_VIDEOQUALITY quality_limit;
    CinemoVideoPropertiesFlags flags;
    CinemoRect source;
    CinemoRect target;
    char device_params[256];
} CinemoVideoProperties;
```

```
typedef struct {
    uchar stats_layer : 1;
    uchar relative_source : 1;
    uchar disable_cropping : 1;
    uchar disable_overscan : 1;
    uchar auto_target : 1;
    uchar disable_upscaling : 1;
    uchar freeze : 1;
    uchar reserved[3];
} CinemoVideoPropertiesFlags;
```

- **zorder**

Specify the base z-order number for the video playback. Only used when multiple players use the same video device. Higher number is on top. Default value should be zero.

- **aspect_mode**

Specify the video aspect ratio mode, which controls the way video will appear in a target display rectangle. Unspecified (zero) value falls back to *CINEMO_OPTION_VIDEO_ASPECT* option. See *CINEMO_OPTION_VIDEO_ASPECT* for more details.

- **quality_limit**

Specify the default video decoding quality level. Unspecified (zero) value will decode all frames.

- **flags.stats_layer**

Show a small statistics window in the top-left corner of the video. (For testing only)

- **flags.relative_source**

Specify source rectangle as relative coordinates (0..1000) instead of actual video pixels. Example left=250, top=250, right=750, bottom=750 would show the center quarter of the video.

- **flags.disable_cropping**

Ignore the encoded video cropping parameters and always show the full video. (For testing only)

- **flags.disable_overscan**

Ignore the encoded video overscan cropping parameters. Example show original 720 pixels wide MPEG-2 content instead of cropped 704 pixels.

- **flags.auto_target**

Use the full window/screen for video playback viewport instead of the manually assigned **target** rectangle.

- **flags.disable_upscaling**

Don't upscale small resolution video, instead show centered in viewport area.

- **flags.freeze**

Temporarily freeze current video frame and don't show newly decoded frames.

- **source**

Specify the source rectangle of the video that should be fitted in the viewport according to the aspect_mode. Depending on flags.relative_source unit is video pixels or relative coordinates (0..1000)

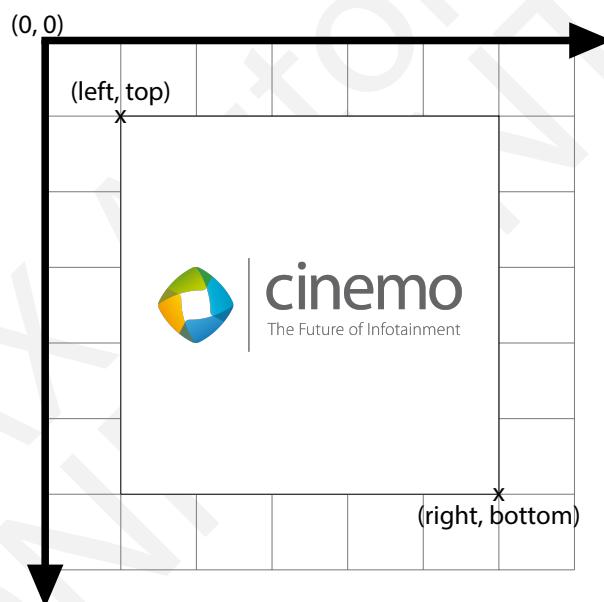
- **target**

Specify the viewport area of window/screen where the video should be fitted according to the aspect_mode. Depending on aspect ratio mode the video might not fill the full viewport, but get additional black borders.

- **device_params**

Is used to pass runtime specific parameters to the video device.

Please note the definition of coordinates in Cinemo:



3.19.8 GetVideoParams

Get the current video device parameters.

Syntax

```
CinemoError GetVideoParams(  
    [out] CinemoVideoParams& params  
) ;
```

Parameters

- **params**

Video device parameter structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.19.9 SetDistributed

Set distributed playback parameters.

Syntax

```
CinemoError SetDistributed(  
    [in]      const CinemoDistributed& params  
) ;
```

Parameters

- **params**

Distributed playback parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is used to enable or disable distributed playback. If enabled, the [ICinemoPlayer](#) instance will be visible on the network as a distributed playback master. If disabled, then all existing distributed playback connections will be terminated.

If [SetDistributed\(\)](#) is called with distributed playback enabled, metadata stored in the playlist for the currently playing track will be re-evaluated for use by the distributed playback discovery service. For further details see [ICinemoPlaylist::GetMetapool\(\)](#).

3.19.10 SetSpeed

Set playback speed.

Syntax

```
CinemoError SetSpeed(  
    [in]     sint32 speed  
) ;
```

Parameters

- **speed**

Playback speed in units of 1000.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The playback speed may be negative or positive. The speed for normal forward playback is 1000. This is also the default state. Values less than -128000 or greater than +128000 are not supported. A running playback graph depends also on other attributes like domain state and cueing.

3.19.11 SetScanTime

Set the scan time.

Syntax

```
CinemoError SetScanTime(  
    [in]      uint32 scan_ms  
) ;
```

Parameters

- **scan_ms**

Scan time in milliseconds.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is used to support scan mode. In this mode, after playing a track for the specified number of milliseconds, playback will skip to the next track. A value of zero will disable scan mode.

3.19.12 SetTimeEventsInterval

Set the interval between periodic *CINEMO_EC_TIME* events.

Syntax

```
CinemoError SetTimeEventsInterval(
    [in]      uint32 time_events_ms
);
```

Parameters

- **time_events_ms**

The interval, in milliseconds, between periodic *CINEMO_EC_TIME* events.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Note that the event is also posted aperiodically, such as when playback is paused or resumed.

3.19.13 SetAudioCaptureCallback

Set an application defined callback function to receive PCM audio samples of the track currently being played.

Syntax

```
CinemoError SetAudioCaptureCallback(
    [in]    void * puser,
    [in]    CinemoCaptureCallback pcall,
    [in]    const CinemoAudioFormat * pformat
    [in]    uint32 flags
);
```

Parameters

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function. If set to NULL, the capturing will be deactivated.

- **pformat**

Specify the audio format in which the callback function should receive PCM samples. If set to NULL, the PCM samples will have the same format as output by the audio decoder. If not NULL, all non-zero fields will override the audio decoder's format fields and a resampling will be performed before passing the samples to the callback function. This resampling will not alter the samples rendered to the audio device. It is not required to specify the blockalign and bitrate fields of the audio format.

- **flags**

Flags can be specified to control the behavior in case that the current playback speed is changed.

- CINEMO_AUDIO_CAPTURE_FLAG_DEFAULT: Do not follow the current playback speed.
- CINEMO_AUDIO_CAPTURE_FLAG_RESAMPLE_TO_PLAYSPEED: If set, resampling will be applied and the captured PCM samples will match the current playback speed.
- CINEMO_AUDIO_CAPTURE_FLAG_USE_TIMESTRETCH: If set, resampling is performed preserving the pitch. For small, frequent speed adjustments, it is recommended to not enable timestretch.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The user defined application callback function will be called synchronously at the audio rendering stage. It is therefore strongly advised to return from the callback function as quickly as possible to prevent a potential underrun of the audio buffer.

To stop receiving PCM samples, a NULL callback function must be specified.

3.19.14 OpenTrack

Open a track, but do not start playback.

Syntax

```
CinemoError OpenTrack(  
    [in]      uint64 track_id  
) ;
```

Parameters

- **track_id**

Track identifier to open.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method opens the track specified by the track ID in the playlist which created this [ICinemoPlayer](#) interface. Note, the track ID differs from the track index. The track ID identifies the track uniquely within the playlist, whereas the track index indicates the current position of the track within the playlist.

The [OpenTrack\(\)](#) command closes currently open track and starts asynchronous opening of new track. It will return immediately once the currently open track is closed. The current playback speed is reset to 1000 (1x). The calling application will be notified when the track is asynchronously opened by means of the *CINEMO_EC_OPEN* event. In the meantime (i.e. between calling [OpenTrack\(\)](#) and receiving the *CINEMO_EC_OPEN* event) most [ICinemoPlayer](#) functions will return *CinemoErrorState*.

3.19.15 CloseTrack

Close the currently open track, release resources and reset attributes to initial state.

Syntax

```
CinemoError CloseTrack();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method doesn't wait for closing the track. If the method succeeds then *CINEMO_EC_CLOSE* event is posted when resources are freed. If this method is called again before a previous [CloseTrack\(\)](#) finished freeing the resources then *CinemoErrorPending* is returned.

3.19.16 PlayTrack

Open a track, and automatically start playback.

Syntax

```
CinemoError PlayTrack(  
    [in]      uint64 track_id  
) ;
```

Parameters

- **track_id**

Track identifier to start playback.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is similar to [OpenTrack\(\)](#) but current play speed not changed and a successfully open (i.e. when *CINEMO_EC_OPEN* event is posted with *CinemoNoError*) is followed by an automatic start of playback from the beginning or end of the track depending on current play speed being positive or negative.

The behavior after reaching the end of the track depends on the current play order and repeat mode of the playlist, which may be set dynamically while the track is playing.

The [PlayTrack\(\)](#) command closes currently open track and starts asynchronous opening of new track same way as [OpenTrack\(\)](#). It will return immediately once the currently open track is closed. The calling application will be notified when the track is asynchronously opened by means of the *CINEMO_EC_OPEN* event. In the meantime (i.e. between calling [PlayTrack\(\)](#) and receiving the *CINEMO_EC_OPEN* event) most [ICinemoPlayer](#) functions will return *CinemoErrorState*.

3.19.17 NextTrack

Skip to the next track.

Syntax

```
CinemoError NextTrack();
```

Return value

If the method succeeds, it returns *CinemoNoError*. If there is no next track, the method returns *CinemoErrorNoMoreItems*. Otherwise, it returns a *CinemoError* code.

Remarks

The [NextTrack\(\)](#) command closes currently open track and starts asynchronous opening of new track same way as [OpenTrack\(\)](#). It will return immediately once the currently open track is closed. The calling application will be notified when the next track is asynchronously opened by means of the *CINEMO_EC_OPEN* event. In the meantime (i.e. between calling [NextTrack\(\)](#) and receiving the *CINEMO_EC_OPEN* event) most [ICinemoPlayer](#) functions will return *CinemoErrorState*.

3.19.18 PreviousTrack

Skip to the previous track.

Syntax

```
CinemoError PreviousTrack();
```

Return value

If the method succeeds, it returns *CinemoNoError*. If there is no previous track, the method returns *CinemoErrorNoMoreItems*. Otherwise, it returns a *CinemoError* code.

Remarks

The [PreviousTrack\(\)](#) command closes currently open track and starts asynchronous opening of new track same way as [OpenTrack\(\)](#). It will return immediately once the currently open track is closed. The calling application will be notified when the previous track is asynchronously opened by means of the *CINEMO_EC_OPEN* event. In the meantime (i.e. between calling [PreviousTrack\(\)](#) and receiving the *CINEMO_EC_OPEN* event) most [ICinemoPlayer](#) functions will return *CinemoErrorState*.

3.19.19 Play

Play the current track (if in stopped state) and reset play speed to 1000 (1x) unless track is remote player like iAP and AVRCP.

Syntax

```
CinemoError Play();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the track is not already playing, this method will start playback of the current track from the beginning, with a forward speed of 1000. With iAP and AVRCP devices this function only starts audio streaming, play speed won't change.

3.19.20 Stop

Stop playback of the current track.

Syntax

```
CinemoError Stop();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.19.21 Seek

Seek to a time position within the current track.

Syntax

```
CinemoError Seek(  
    [in]      const CinemoPosition& time  
) ;
```

Parameters

- **time**

Seek time position in various units.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the track is not already playing, playback will be started. If the seek position is negative, playback will start at the beginning of the track. If the seek position exceeds the track duration, playback will start at the end of the track.

3.19.22 SeekTitleChapter

Seek to a title or chapter.

Syntax

```
CinemoError SeekTitleChapter(  
    [in]      uint32 title,  
    [in]      uint32 chapter  
) ;
```

Parameters

- **title**

Title number.

- **chapter**

Chapter number.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The value of title number might be a specific title index, or zero to seek to a chapter inside the current title. When seeking to a specific title the value of chapter number can be zero to seek to the default chapter of the title. This method only succeeds for tracks which contain multiple titles or chapters, for example DVD-Video discs, VCD/SVCD discs, or audio books.

3.19.23 ShowMenu

Show a menu.

Syntax

```
CinemoError ShowMenu(  
    [in]      CINEMO_MENU id  
) ;
```

Parameters

- **id**

Menu identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for tracks which contain menus, for example DVD-Video discs. The menu identifier may be one of the following values:

- *CINEMO_MENU_TOP*
- *CINEMO_MENU_TITLE*
- *CINEMO_MENU_ROOT*
- *CINEMO_MENU_SUBPICTURE*
- *CINEMO_MENU_AUDIO*
- *CINEMO_MENU_ANGLE*
- *CINEMO_MENU_CHAPTER*
- *CINEMO_MENU_POPUP*

3.19.24 ResumeTitle

Resume title playback (after Stop or ShowMenu).

Syntax

```
CinemoError ResumeTitle();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

When [Stop\(\)](#) or [ShowMenu\(\)](#) is executed, the current playback position is saved. This method will restore the saved playback position.

3.19.25 ReturnFromSubmenu

Navigate up through menu hierarchy.

Syntax

```
CinemoError ReturnFromSubmenu();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is implemented in accordance with the DVD specifications for “GO_UP” user operations, however it is unlikely to be called by a typical player application.

3.19.26 NextChapter

Skip to the next chapter.

Syntax

```
CinemoError NextChapter();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for tracks which contain multiple chapters, for example DVD-Video discs, VCD/SVCD discs, or audio books.

3.19.27 PreviousChapter

Skip to the previous chapter.

Syntax

```
CinemoError PreviousChapter();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for tracks which contain multiple chapters, for example DVD-Video discs, VCD/SVCD discs, or audio books.

3.19.28 ReplayChapter

Replay current chapter from beginning.

Syntax

```
CinemoError ReplayChapter();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for tracks which contain multiple chapters, for example DVD-Video discs, VCD/SVCD discs, or audio books.

3.19.29 SetRepeatChapter

Set repeat mode of the chapter.

Syntax

```
CinemoError SetRepeatChapter(  
    [in]      CINEMO_REPEAT CHAPTER repeat  
) ;
```

Parameters

- **repeat**

The chapter repeat mode.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for DVD-Video discs. The repeat mode is set in the scope of the player and valid until changed by a subsequent call of this method. This method does not check for prohibited user operations. A chapter is not repeated if the related user operation is prohibited.

3.19.30 SetRepeatTitle

Set repeat mode of the title.

Syntax

```
CinemoError SetRepeatTitle(  
    [in]      CINEMO_REPEAT_TITLE repeat  
) ;
```

Parameters

- **repeat**

The title repeat mode.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for DVD-Video discs. The repeat mode is set in the scope of the player and valid until changed by a subsequent call of this method. This method does not check for prohibited user operations. A title is not repeated if the related user operation is prohibited.

3.19.31 StepForward

Step forward one frame.

Syntax

```
CinemoError StepForward();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the play speed is not currently zero, this method will have the same effect as SetSpeed(0), to pause playback. Otherwise, playback will advance by a single video frame.

3.19.32 StepBackward

Step backward one frame.

Syntax

```
CinemoError StepBackward();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the play speed is not currently zero, this method will have the same effect as *SetSpeed(0)*, to pause playback. Otherwise, playback will step backward a single video frame.

3.19.33 SelectAudio

Select an audio stream.

Syntax

```
CinemoError SelectAudio(  
    [in]      uint32 id  
) ;
```

Parameters

- **id**

Audio stream identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The audio stream identifier may be either zero, or the one-based index of the audio stream in the current track. If set to zero, audio output will be disabled (i.e. no audio stream will be selected).

3.19.34 SelectAngle

Select an angle or video stream.

Syntax

```
CinemoError SelectAngle(  
    [in]      uint32 id  
) ;
```

Parameters

- **id**

Angle or video stream identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The angle identifier may be either zero, or the one-based index of the angle or video stream in the current track. If set to zero, video output will be disabled. Angles may refer to the multi-angle properties of some DVD-Video titles, in all other cases angles refer to video streams.

3.19.35 SelectSubpicture

Select a subpicture stream.

Syntax

```
CinemoError SelectSubpicture(  
    [in]      uint32 id  
) ;
```

Parameters

- **id**

Subpicture stream identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The subpicture stream identifier may be either zero, or the one-based index of the subpicture stream in the current track. If set to zero, subpicture output will be disabled (i.e. no subpicture stream will be selected).

3.19.36 AddExternalSubtitle

Add subtitles loaded from an external resource.

Syntax

```
CinemoError AddExternalSubtitle(  
    [in]      const char * szurl  
) ;
```

Parameters

- **szurl**
URL identifying the external subtitle.

Return value

If the method succeeds, it returns *CinemoNoError*. If there is no video stream available, the method returns *NmeErrorStreamUnavailable*. Otherwise, it returns a *CinemoError* code.

Remarks

This feature is controlled by the *CINEMO_OPTION_SUBTITLE_ENABLE* configuration option.

The supported subtitle formats are:

- AQTITLE (AQT),
- Cheetah (ASC),
- Lyrics (LRC),
- MicroDVD (TXT),
- Mplayer(2) (MPL),
- RealText (RT),
- Sasami Script (S2K),
- SAMI Captioning (SAMI, SMI),
- SubRip (SRT),
- SubStation Alpha (SSA, ASS),
- SubViewer (SUB),
- 3GPP TS 26.245 V6.0.0 (TTXT),
- Ulead Subtitle,
- Universal Subtitle Format (USF),
- VobSub (IDX).

The format is detected independently of the file extension. The method loads and adds all subtitle streams contained in the resource.

HUMAX Confidential

3.19.37 SelectButton

Select a button.

Syntax

```
CinemoError SelectButton(  
    [in]      uint32 id  
) ;
```

Parameters

- **id**

Button identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The button identifier is the one-based index of the button within the current set of buttons, for example, in a DVD-Video menu. It is not allowed to select an invalid button.

3.19.38 ActionButton

Action currently selected button.

Syntax

```
CinemoError ActionButton();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will action the currently selected button, for example, in a DVD-Video menu. This means that its navigation command will be executed.

3.19.39 SelectButtonRelative

Select button relative to the currently selected button.

Syntax

```
CinemoError SelectButtonRelative(  
    [in]      const CinemoPoint& point  
) ;
```

Parameters

- **point**

Specify the relative XY direction of the button to select.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is intended for the navigation of DVD menus by arrow keys. For example, to move the selection up, the relative XY coordinates should be specified as $(0, -1)$. Similarly, the parameter for down is $(0, 1)$; left is $(-1, 0)$; and right is $(1, 0)$.

3.19.40 SelectButtonPosition

Select button at XY position.

Syntax

```
CinemoError SelectButtonPosition(  
    [in]      const CinemoPoint& point  
) ;
```

Parameters

- **point**

Specify the XY coordinates of the button.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is intended for the navigation of DVD menus by mouse cursor, and implements hover function. To activate the button when a mouse button is clicked, [ActionButtonPosition\(\)](#) should be called instead.

3.19.41 ActionButtonPosition

Action button at XY position.

Syntax

```
CinemoError ActionButtonPosition(  
    [in]      const CinemoPoint& point  
) ;
```

Parameters

- **point**

Specify the XY coordinates of the button.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is intended for the navigation of DVD menus. With Blu-ray discs this function simulates *CINEMO_MOUSEEVENT_LBUTTONDOWN* and *CINEMO_MOUSEEVENT_LBUTTONUP* mouse events. It's recommended to use [ICinemoPlayer2::PostMouseEvent\(\)](#) method instead to send mouse events.

3.19.42 AcceptParentalLevel

Reply to currently pending temporary parental level change request.

Syntax

```
CinemoError AcceptParentalLevel(  
    [in]      uint32 accept  
) ;
```

Parameters

- **accept**

Specify whether the temporary parental level change is accepted. Set to a non-zero value to accept the temporary parental level change. Set to zero otherwise.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is intended for DVD playback. By navigation command, DVDs may request a temporary parental level change. If this feature is enabled via the option *CINEMO_OPTION_DVD_PARENTAL_LEVEL_CHANGE*, upon encountering a request for a temporary parental level change in the DVD navigator, an event *CINEMO_EC_PARENTAL_LEVEL* will be generated and sent to the application. The DVD navigator will then wait for the reply to the temporary parental level change request, which is implemented using the given method.

3.19.43 AcceptCMI

Reply to currently pending content management information (CMI) change request. The request is issued by means of the event [CINEMO_EC_CMI](#).

Syntax

```
CinemoError AcceptCMI(  
    [in]      uint32 accepted_cmi  
) ;
```

Parameters

- **accepted_cmi**

The CMI flags that are accepted for rendering and output. The required CMI flags are passed by the event [CINEMO_EC_CMI](#).

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If this feature is enabled via the option *CINEMO_OPTION_PROTECTION_ENABLE_ACCEPT_CMI*, upon encountering a change in the CMI of the currently rendered content, a *CINEMO_EC_CMI* event will be issued to the application containing the requested CMI flags. The rendering will be disabled until the requested CMI is accepted by the application using the given method.

The extraction of CMI information is currently only supported for CSS protected DVD-Video.

3.19.44 GetStatus

Get the current playback status.

Syntax

```
CinemoError GetStatus (
    [out]    CinemoStatus& status
);
```

Parameters

- **status**

For returning the current playback status.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a *CinemoStatus* structure will be initialized with a snapshot of the current playback status. Note there is no information in this structure which could not also be known by an application monitoring playback events.

The CinemoStatus structure is defined here:

```

typedef struct {
    uint64 track;           /* track identifier */
    CINEMO_STATUS status;   /* CINEMO_STATUS */
    CINEMO_DOMAIN domain;  /* CINEMO_DOMAIN */
    uint32 cue;             /* cue status */
    sint32 speed;           /* speed */
    uint32 puops;           /* prohibited user operations */
    uint32 still;            /* still mode */
    uint32 scan_ms;          /* scantime in milliseconds */
    uint32 time_events_ms;   /* time events interval in milliseconds */
    uint32 popup_menu;       /* popup menu state (BD specific) */
    uint32 title_transition; /* title transition state */
    uint32 audio_watermark_mute; /* audio watermark mute state */
    uint32 streaming;        /* device is streaming audio (AVRCP nav specific) */
    CINEMO_REPEAT CHAPTER repeat_chapter; /* repeat chapter state */
    CINEMO_REPEAT TITLE repeat_title;      /* repeat title state */
    struct {
        uint32 id;           /* current title */
        uint32 count;         /* total count of titles */
    } title;
    struct {
        uint32 id;           /* current chapter */
        uint32 count;         /* total count of chapters */
    } chapter;
    struct {
        uint32 id;           /* current audio stream */
        uint32 count;         /* total count of audio streams */
    } audio;
    struct {
        uint32 id;           /* current angle */
        uint32 count;         /* total count of angles */
    } angle;
    struct {
        uint32 id;           /* current subpicture stream */
        uint32 count;         /* total count of subpicture streams */
    } subpicture;
    struct {
        uint32 id;           /* current secondary video stream */
        uint32 count;         /* total count of secondary video streams */
    } secondary_video;
    struct {
        uint32 id;           /* current secondary audio stream */
        uint32 count;         /* total count of secondary audio streams */
    } secondary_audio;
    struct {
        uint32 id;           /* current button */
        uint32 count;         /* total count of buttons */
    } button;
} CinemoStatus;

```

3.19.45 GetGraphStatus

Get current decoder graph status.

Syntax

```
CinemoError GetGraphStatus(
    [out]    CinemoGraphStatus& status
);
```

Parameters

- **status**

For returning the current graph status.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a *CinemoGraphStatus* structure will be initialized with a snapshot of the current playback graph status. *CinemoGraphStatus* structure is defined as follows:

```
typedef struct {
    struct {
        int alive;           /* graph is playing */
        int speed;          /* graph speed */
        int audio_queued;   /* audio queued bytes */
        int video_queued;   /* video queued bytes */
        int total_queued;   /* total queued bytes */
        int clients;         /* number of clients (also counting local playback) */
    } graph;
    struct {
        int exists;          /* audio exists */
        int channels;        /* channels */
        int channelconfig;   /* channel configuration mask */
        int samplerate;      /* samples/sec */
        int bits;             /* bits/sample */
        int byterate;         /* bytes/second */
        int flags;            /* CINEMO_AUDIOFLAG_xxx */
    } audio;
    struct {
        int exists;          /* video exists */
        int cx;               /* width after cropping */
        int cy;               /* height after cropping */
        int framerate;        /* frames/second x 1000 */
        int interlaced;       /* interlaced */
        int byterate;         /* bytes/second */
        int hw_decode;        /* hardware decoding active */
        int quality;          /* quality control state */
    } video;
}
```

```
struct {
    int exists;           /* parse status exists */
    int bytes;            /* total bytes */
    int packets;          /* total packets */
    int error_packets;   /* total packet errors */
} parse;
struct {
    int clock_ms;         /* clock */
    int audio_ms;         /* audio device */
} drift;
char audio_codec[64];      /* audio codec name */
char video_codec[64];      /* video codec name */
} CinemoGraphStatus;
```

HUMAX AUTOMOTIVE CONFIDENTIAL

3.19.46 GetQualityInfo

Get current audio and video renderer quality info.

Syntax

```
CinemoError GetQualityInfo(
    [out]    CinemoQualityInfo& info
);
```

Parameters

- **info**

For returning quality information.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a *CinemoQualityInfo* structure will be initialized with a snapshot of the current audio and video quality statistics.

The *CinemoQualityInfo* structure is defined here:

```
typedef struct {
    uint64 decoded_samples;           /* decoded sample count */
    uint32 crc32;                   /* crc32 for all decoded samples */
} CinemoAudioQualityInfo;

typedef struct {
    uint64 decoded_samples;           /* decoded sample count */
    uint64 dropped_samples;          /* dropped sample count */
    uint32 crc32;                   /* crc32 for all decoded samples */
} CinemoVideoQualityInfo;

typedef struct {
    uint32 cpu_usage;                /* 1000 = 1% */
    CinemoAudioQualityInfo audio;    /* audio quality info */
    CinemoVideoQualityInfo video;    /* video quality info */
} CinemoQualityInfo;
```

3.19.47 GetPosition

Get play position within current track.

Syntax

```
CinemoError GetPosition(  
    [in/out] CinemoPosition& time  
) ;
```

Parameters

- **time**

Returns the current playback position.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method returns the current playback position in units specified in the *CinemoPosition* structure. If an application wishes the position to be returned in specific units, it should assign these units in the structure *before* calling this method.

3.19.48 GetDuration

Get duration of current track.

Syntax

```
CinemoError GetDuration(  
    [in/out] CinemoPosition& time  
) ;
```

Parameters

- **time**

Returns the current track duration.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method returns the current track duration in units specified in the *CinemoPosition* structure. If an application wishes the duration to be returned in specific units, it should assign these units in the structure *before* calling this method.

3.19.49 GetBuffered

Get buffer status of the current track.

Syntax

```
CinemoError GetBuffered(  
    [out] CinemoBuffered& info  
) ;
```

Parameters

- **info**

Returns the current buffer status.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a *CinemoBuffered* structure will be initialized with a snapshot of the current buffer status. The buffer status can be used, for example, to draw a seek slider indicating which parts of a track have been buffered or downloaded.

The *CinemoBuffered* structure is defined here:

```
typedef struct {  
    sint64 T1;  
    sint64 T2;  
    uint64 fillrate;          /* in bytes/second */  
    uint32 curbytes;  
    uint32 maxbytes;  
} CinemoBuffered;
```

3.19.50 GetAudio

Get audio stream attributes.

Syntax

```
CinemoError GetAudio(  
    [in]  uint32 id,  
    [out] CinemoMediaType& media  
) ;
```

Parameters

- **id**
Stream identifier.
- **media**
Returned audio stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The audio stream identifier is the one-based index of the audio stream in the current track. If the function succeeds, the returned media type will completely describe this stream.

3.19.51 GetAngle

Get angle or video stream attributes.

Syntax

```
CinemoError GetAngle(  
    [in]  uint32 id,  
    [out] CinemoMediaType& media  
) ;
```

Parameters

- **id**

Stream identifier.

- **media**

Returned video stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The angle or video stream identifier is the one-based index of the video stream in the current track. If the function succeeds, the returned media type will completely describe this stream.

3.19.52 GetSubpicture

Get subpicture attributes.

Syntax

```
CinemoError GetSubpicture(
    [in]  uint32 id,
    [out] CinemoMediaType& media
);
```

Parameters

- **id**
Stream identifier.
- **media**
Returned subpicture stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The subpicture stream identifier is the one-based index of the subpicture stream in the current track. If the function succeeds, the returned media type will completely describe this stream.

3.19.53 GetMediaInfo

Get information about the medium.

Syntax

```
CinemoError GetMediaInfo(  
    [out] CinemoMediaInfo& mediainfo  
) ;
```

Parameters

- **mediainfo**

media information structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method takes a *CinemoMediaInfo* structure that will be filled with information about the current medium. The *CinemoMediaInfo* structure is defined here:

```
typedef struct {  
    uint32 num_titles;           /* number of titles on the medium */  
    uint32 num_dvdvr_programs;   /* number of original programs (DVD-VR only) */  
    uint32 encrypted;           /* encryption information */  
} CinemoMediaInfo;
```

The *num_dvdvr_programs* field is only set for DVD-VR discs, where it holds the number of original programs.

3.19.54 GetTitle

Get information about a title.

Syntax

```
CinemoError GetTitle(
    [in/out] CinemoTitle& info
);
```

Parameters

- **info**
title information structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method takes a *CinemoTitle* structure that will be filled with information about a given title. The *CinemoTitle* structure is defined here:

```
typedef struct {
    uint32 title_id;                                /* title */
    uint32 num_chapters;                            /* chapters */
    uint32 num_audio;                               /* audio streams */
    uint32 num_angle;                               /* angles */
    uint32 num_subpicture;                          /* subpicture streams */
    uint32 num_audio_unsupported;                  /* unsupported audio streams */
    uint32 num_angle_unsupported;                  /* unsupported angles */
    uint32 num_subpicture_unsupported;            /* unsupported subpicture streams */
    uint32 num_subpicture_styles;                 /* subpicture styles (BD nav specific) */
    uint32 num_secondary_video;                   /* secondary video streams (BD nav specific) */
    uint32 num_secondary_audio;                   /* secondary audio streams (BD nav specific) */
    CinemoTitleFlags flags;                        /* title type and permissions (BD nav specific) */
    uint32 reserved;                               /* reserved */
    pts70mhz duration;                            /* 70.56Mhz duration */
} CinemoTitle;
```

You will need to assign a value to the *title_id* field before calling this function to specify the title you want to get information from. The value of this field might be a specific title index, or 0 to get information on the current title.

3.19.55 GetChapter

Get information about a chapter.

Syntax

```
CinemoError GetChapter(  
    [in/out] CinemoChapter& info  
) ;
```

Parameters

- **info**
chapter information structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method takes a *CinemoChapter* structure that will be filled with information about a given chapter. The *CinemoChapter* structure is defined here:

```
typedef struct {  
    uint32 title_id;                      /* title */  
    uint32 chapter_id;                     /* chapter */  
    pts70mhz start;                       /* 70.56Mhz offset */  
    pts70mhz duration;                    /* 70.56Mhz duration */  
} CinemoChapter;
```

You will need to assign a value to the *title_id* and *chapter_id* fields before calling this function to specify the chapter you want to get information from and its related title. The value of these fields might be a specific title and chapter index, or 0 to get information on the current chapter of the current title.

3.19.56 SetSessionData

Set session data for the distributed playback server or client.

Syntax

```
CinemoError SetSessionData(  
    [in]  struct ICinemoMetapool * ppool  
) ;
```

Parameters

- **ppool**

Pointer to [ICinemoMetapool](#) object containing session data.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method allows the caller to set application specific metadata for the current distributed playback session. The metadata will be visible to all [ICinemoPlayer](#) instances connected to the same distributed playback session (i.e. on the server and all connected clients).

The [ICinemoMetapool](#) object passed to this method may be NULL, or otherwise, may contain any application-defined metadata. Cinemo does not parse or care about the contents, and will only forward the metadata to all connected clients. The [ICinemoMetapool](#) interface may be released after calling this method.

After calling [SetSessionData\(\)](#), the *CINEMO_EC_SESSION_DATA* event will be posted on the distributed playback server, and all connected clients, to indicate that the distributed playback session metadata has changed. The [GetSessionPool\(\)](#) method may be used on the server, and all connected clients, to retrieve the metadata.

3.19.57 GetSessionPool

Get the distributed playback session pool.

Syntax

```
CinemoError GetSessionPool(  
    [out]    struct ICinemoMetapool ** pppool  
) ;
```

Parameters

- **pppool**

The address of a pointer to receive the requested [ICinemoMetapool](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [GetSessionPool\(\)](#) method will return a copy of the current distributed playback session metadata pool. The returned [ICinemoMetapool](#) interface may be used to enumerate the application specific metadata previously written by [SetSessionData\(\)](#) on the distributed playback server.

The reference count of the returned [ICinemoMetapool](#) interface will be automatically incremented. Its [ICinemoUnknown::Release\(\)](#) method should be called when it is no longer required.

3.19.58 InitMetapool

Obtain an [ICinemoMetapool](#) interface for reading metadata.

Syntax

```
CinemoError InitMetapool(  
    [out]    struct ICinemoMetapool ** pp  
) ;
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoMetapool](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The reference count of the returned [ICinemoMetapool](#) interface will be automatically incremented. Its [ICinemoUnknown::Release\(\)](#) method should be called when it is no longer required.

3.19.59 InitPlaylistMetapool

Obtain an [ICinemoMetapool](#) interface for reading the playlist metadata for the current track.

Syntax

```
CinemoError InitPlaylistMetapool(
    [out]    struct ICinemoMetapool ** pp
);
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoMetapool](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The reference count of the returned [ICinemoMetapool](#) interface will be automatically incremented. Its [ICinemoUnknown::Release\(\)](#) method should be called when it is no longer required.

3.19.60 InitPainter

Obtain an [ICinemoPainter](#) interface for supporting painting by the GUI framework thread. This feature is platform dependent.

Syntax

```
CinemoError InitPainter(  
    [out]    struct ICinemoPainter ** pp  
) ;
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoPainter](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The reference count of the returned [ICinemoPainter](#) interface will be automatically incremented. Its [ICinemoUnknown::Release\(\)](#) method should be called when it is no longer required.

3.19.61 SaveState

Save current playback state.

Syntax

```
CinemoError SaveState (
    [out]    struct ICinemoBlob ** pp
);
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoBlob](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will save the current playback state to a ‘blob’. The state may be subsequently restored by means of the [RestoreState\(\)](#) method.

3.19.62 RestoreState

Restore current playback state.

Syntax

```
CinemoError RestoreState (
    [in]     const void * pdata,
    [in]     uint32 nbytes
);
```

Parameters

- **pdata**

The address of a ‘blob’ of data.

- **nbytes**

The size of the ‘blob’ in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will restore the playback state previously saved by means of the [SaveState\(\)](#) method. It is required that the track for which the state is intended be currently open, so the correct sequence of calls to restore a playback state is:

1. [OpenTrack\(\)](#)
2. Wait for the *CINEMO_EC_OPEN* indicating *CinemoNoError*
3. [RestoreState\(\)](#)

3.19.63 SignalPlaylistChanged

Signal that the parent playlist has changed.

Syntax

```
CinemoError SignalPlaylistChanged();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method should only be called when using [ICinemoPlayer](#) in standalone mode, that is, outside of an [ICinemoPlaylist](#). Otherwise, this method should never be called.

This method is intended to support dynamic playlists (playlists which may change during playback). If the tracks in a playlist change, this method should be called on all attached players. This will cause all players to re-evaluate both their currently playing track and any next or previous track which may be opened and queued for gapless playback.

3.19.64 TakeSnapshot

Take a snapshot of the current video frame.

Syntax

```
CinemoError TakeSnapshot(  
    [in/out]          CinemoThumbFormat& thumb           /* size and format for snapshot */  
) ;
```

Parameters

- **thumb**

- [in] The requested size and format for the snapshot.
- [out] The actual size and format of the created snapshot.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will take a snapshot of the current video frame, encoded with the specified format and scaled to not more than the specified size in pixels. If successful, the result is stored in the metapool under the metaname *CINEMO_METANAME_SNAPS*, and the *CINEMO_EC_METADATA* event is posted.

3.19.65 GetWindowDeviceName

Get window device name string with platform specific window or surface handles as URL parameters.

Syntax

```
CinemoError GetWindowDeviceName (
    [out]    struct ICinemoUTF8 ** pp
);
```

Parameters

- **pp**

The returned device name value.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.19.66 GetTrackURL

Get URL for the currently playing track from the playlist or track provider.

Syntax

```
CinemoError GetTrackURL(  
    [out]    struct ICinemoUTF8 ** pp  
) ;
```

Parameters

- **pp**

The returned URL.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.19.67 GetTrackVFSAttributes

Get VFS attributes for the currently playing track.

Syntax

```
CinemoError GetTrackVFSAttributes (
    [out]    CinemoVFSAttributes& attrs
);
```

Parameters

- **attrs**

The returned VFS attributes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.19.68 GetDistributedServerURL

Get the URL of the Cinemo Distributed Playback Server.

Syntax

```
CinemoError GetDistributedServerURL(  
    [out]    struct ICinemoUTF8 ** pp  
) ;
```

Parameters

- **pp**

The returned URL.

Return value

If the method succeeds, it returns *CinemoNoError*. It returns *CinemoFalse* if the distributed server is disabled. Otherwise, it returns a *CinemoError* code.

Remarks

This method returns the URL of the distributed server.

3.20 ICinemoPlayer2

The [ICinemoPlayer2](#) interface is an alternative to the [ICinemoPlayer](#) which supports playback of all files and media types supported by [ICinemoPlayer](#), with support for the additional features required for Blu-ray and iAP playback.

NOTE: In this version of this document the documentation of the [ICinemoPlayer2](#) interface functions in the following sections is still preliminary and should be considered with care.

This thread-safe interface may be obtained in two ways:

- **Playlist mode**

Call [ICinemoPlaylist::InitPlayback2\(\)](#) on an instantiated playlist. First it is necessary to create an [ICinemoPlaylist](#). There is no limit to the number of players which may be instantiated from a single playlist. Each player maintains its independent playback state, so it is possible to play multiple tracks from the same playlist at the same time.

- **Standalone mode**

Call [CinemoCreatePlayer2\(\)](#) with an [ICinemoTrackProvider](#) interface provided by the calling application. In this case, the application is responsible for maintaining tracks. In all other respects, the operation of the player is the same.

ICinemoPlayer2 Methods

METHOD	DESCRIPTION
SetEventCallback	Set callback function to receive player events
SetEventQueue	Set event queue to receive playlist events
SetEventSourceID	Set source ID for posted events
SetGraphParams	Set graph creation parameters
SetAudioParams	Set audio device parameters
GetAudioParams	Get current audio device parameters
SetVideoParams	Set video device parameters
GetVideoParams	Get current video device parameters
SetDistributed	Set distributed playback parameters
SetScanTime	Set the scan time
SetTimeEventsInterval	Set the interval between periodic CINEMO_EC_TIME events
OpenTrack	Open a track, but do not start playback
PlayTrack	Open a track, and automatically start playback
CloseTrack	Close the current track
NextTrack	Skip to the next track
PreviousTrack	Skip to the previous track
Play	Play the current track (if in stopped state)
Seek	Seek to a time position within current track

METHOD	DESCRIPTION
<u>SeekTitleChapter</u>	Seek to a title or chapter
<u>ShowMenu</u>	Show menu
<u>ResumeTitle</u>	Resume title playback (after Stop or ShowMenu)
<u>ReturnFromSubmenu</u>	Navigate up through menu hierarchy
<u>ReplayChapter</u>	Replay current chapter from beginning
<u>SetRepeatChapter</u>	Set repeat mode of the chapter
<u>SetRepeatTitle</u>	Set repeat mode of the title
<u>StepForward</u>	Step forward one frame
<u>StepBackward</u>	Step backward one frame
<u>SelectAudio</u>	Select an audio stream
<u>SelectAngle</u>	Select an angle or video stream
<u>SelectSubpicture</u>	Select a subpicture stream
<u>AddExternalSubtitle</u>	Add subtitles loaded from an external resource
<u>GetStatus</u>	Get current playback status
<u>GetGraphStatus</u>	Get current decoder graph status
<u>GetQualityInfo</u>	Get current audio and video renderer quality info
<u>GetPosition</u>	Get play position within current track
<u>GetDuration</u>	Get duration of current track
<u>GetBuffered</u>	Get buffer status of the current track
<u>GetAudio</u>	Get audio stream attributes
<u>GetAngle</u>	Get angle or video stream attributes
<u>GetSubpicture</u>	Get subpicture attributes
<u>GetMediaInfo</u>	Get information about the medium
<u>GetTitle</u>	Get information about a title
<u>GetTitleFlags</u>	Get flag only information about a title
<u>GetChapter</u>	Get information about a chapter
<u>SetSessionData</u>	Set session data for the distributed playback server or client
<u>GetSessionPool</u>	Get the distributed playback session pool
<u>InitMetapool</u>	Obtain an <u>ICinemoMetapool</u> interface for reading metadata
<u>InitPlaylistMetapool</u>	Obtain an <u>ICinemoMetapool</u> interface for reading the playlist metadata of the current track
<u>InitPainter</u>	Obtain an <u>ICinemoPainter</u> interface for painting video
<u>SaveState</u>	Save current playback state
<u>RestoreState</u>	Restore current playback state
<u>SignalPlaylistChanged</u>	Signal that the parent playlist has changed

METHOD	DESCRIPTION
<u>TakeSnapshot</u>	Take a snapshot of the current video frame
<u>GetWindowDeviceName</u>	Get window device name string with platform specific window or surface handles as URL parameters.
<u>SelectSubpictureStyle</u>	Select a style for current subpicture stream
<u>SelectSecondaryAudio</u>	Select a secondary audio stream
<u>SelectSecondaryVideo</u>	Select a secondary video stream
<u>GetSecondaryAudio</u>	Get secondary audio stream attributes
<u>GetSecondaryVideo</u>	Get secondary video stream attributes
<u>PostKeyUserEvent</u>	Post user key event to player
<u>PostKeyEvent</u>	Post generic key event to player
<u>PostMouseEvent</u>	Post mouse event to player
<u>GetTrackURL</u>	Get URL for the currently playing track from the playlist or track provider.
<u>GetTrackVFSAttributes</u>	Get VFS attributes for the currently playing track.
<u>GetDistributedServerURL</u>	Get the URL of the Cinemo Distributed Playback Server

3.20.1 SetEventCallback

This method has the same behavior as [ICinemoPlayer::SetEventCallback\(\)](#).

3.20.2 SetEventQueue

This method has the same behavior as [ICinemoPlayer::SetEventQueue\(\)](#).

3.20.3 SetEventSourceID

This method has the same behavior as [ICinemoPlayer::SetEventSourceID\(\)](#).

3.20.4 SetGraphParams

This method has the same behavior as [ICinemoPlayer::SetGraphParams\(\)](#).

3.20.5 SetAudioParams

This method has the same behavior as [ICinemoPlayer::SetAudioParams\(\)](#).

3.20.6 GetAudioParams

This method has the same behavior as [ICinemoPlayer::GetAudioParams\(\)](#).

3.20.7 SetVideoParams

This method has the same behavior as [ICinemoPlayer::SetVideoParams\(\)](#).

3.20.8 GetVideoParams

This method has the same behavior as [ICinemoPlayer::GetVideoParams\(\)](#).

3.20.9 SetDistributed

This method has the same behavior as [ICinemoPlayer::SetDistributed\(\)](#).

3.20.10 SetScanTime

Set the scan time.

Syntax

```
CinemoError SetScanTime(  
    [in]      uint32 scan_ms  
) ;
```

Parameters

- **scan_ms**

Scan time in milliseconds.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is used to support scan mode. In this mode, after playing a track for the specified number of milliseconds, playback will skip to the next track. A value of zero will disable scan mode.

With Blu-ray discs this feature is not supported and a corresponding *CinemoError* code will be returned.

3.20.11 SetTimeEventsInterval

This method has the same behavior as [ICinemoPlayer::SetTimeEventsInterval\(\)](#).

3.20.12 OpenTrack

Open a track, but do not start playback.

Syntax

```
CinemoError OpenTrack(  
    [in]      uint64 track_id  
) ;
```

Parameters

- **track_id**

Track identifier to open.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method opens the track specified by the track ID in the playlist which created this [ICinemoPlayer2](#) interface. Note, the track ID differs from the track index. The track ID identifies the track uniquely within the playlist, whereas the track index indicates the current position of the track within the playlist.

This method is fully asynchronous. It will return immediately. The calling application will be notified when the track is asynchronously opened by means of the *CINEMO_EC_OPEN* event. In the meantime (i.e. between calling [OpenTrack\(\)](#) and receiving the *CINEMO_EC_OPEN* event) most [ICinemoPlayer2](#) methods will return *CinemoErrorState*.

3.20.13 PlayTrack

Open a track, and automatically start playback.

Syntax

```
CinemoError PlayTrack(  
    [in]      uint64 track_id  
) ;
```

Parameters

- **track_id**

Track identifier to start playback.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is equivalent to [OpenTrack\(\)](#) followed by an automatic [Play\(\)](#) command when the track is successfully opened (i.e. when *CINEMO_EC_OPEN* event is posted with *CinemoNoError*).

The behavior on reaching the end of the track depends on the current play order and repeat mode of the playlist, which may be set dynamically while the track is playing.

The [PlayTrack\(\)](#) command is fully asynchronous same way as [OpenTrack\(\)](#). It will return immediately. The calling application will be notified when the track is asynchronously opened by means of the *CINEMO_EC_OPEN* event. In the meantime (i.e. between calling [PlayTrack\(\)](#) and receiving the *CINEMO_EC_OPEN* event) most [ICinemoPlayer](#) functions will return *CinemoErrorState*.

3.20.14 CloseTrack

Close the currently open track and release resources.

Syntax

```
CinemoError CloseTrack();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method doesn't wait for closing the track. *CINEMO_EC_CLOSE* event is posted when resources are freed.

CINEMO_EC_CLOSE with *CinemoErrorAppRunning* argument indicates Blu-ray disc-unbound applications still running in background.

3.20.15 NextTrack

This method has the same behavior as [ICinemoPlayer::NextTrack\(\)](#).

3.20.16 PreviousTrack

This method has the same behavior as [ICinemoPlayer::PreviousTrack\(\)](#).

3.20.17 Play

Play the current track (if in stopped state).

Syntax

```
CinemoError Play();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the track is not already playing, this method will start playback of the current track from the beginning, with a forward speed of 1000. With iAP and AVRCP devices this function only starts audio streaming, play speed won't change. With Blu-ray discs, iAP and AVRCP devices this function is not equivalent to *CINEMO_KEY_PLAY* user key event. It's only needed when [ICinemoPlayer2::OpenTrack\(\)](#) was used instead of [ICinemoPlayer2::PlayTrack\(\)](#).

3.20.18 Seek

Seek to a time position within the current track or title.

Syntax

```
CinemoError Seek(  
    [in]      const CinemoPosition& time  
) ;
```

Parameters

- **time**

Seek time position in various units.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the seek position is negative, playback will start at the beginning of the track. If the seek position exceeds the track duration, playback will start at the end of the track.

3.20.19 SeekTitleChapter

Seek to a title or chapter within the current track.

Syntax

```
CinemoError SeekTitleChapter(  
    [in]      uint32 title,  
    [in]      uint32 chapter  
) ;
```

Parameters

- **title**

Title number.

- **chapter**

Chapter number.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The value of title number might be a specific title index, or zero to seek to a chapter inside the current title. When seeking to a specific title the value of chapter number can be zero to seek to the default chapter of the title.

This method only succeeds for tracks which contain multiple titles or chapters, for example DVD-Video discs, VCD/SVCD discs, or audio books.

With Blu-ray discs seeking to a title and seeking to a chapter are two different user operations. Only one can be selected at a time and one of the two parameters must be zero. For Blu-ray discs additionally to the one-based title numbers there are two special values referring to the first playback title (*CINEMO_TITLE_FIRST_PLAYBACK*) and top menu title (*CINEMO_TITLE_TOP_MENU*).

3.20.20 ShowMenu

Show menu.

Syntax

```
CinemoError ShowMenu(  
    [in]      CINEMO_MENU id  
) ;
```

Parameters

- **id**

Menu identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for tracks which contain menus, for example DVD-Video discs. The menu identifier may be one of the following values:

- *CINEMO_MENU_TOP*
- *CINEMO_MENU_TITLE*
- *CINEMO_MENU_ROOT*
- *CINEMO_MENU_SUBPICTURE*
- *CINEMO_MENU_AUDIO*
- *CINEMO_MENU_ANGLE*
- *CINEMO_MENU_CHAPTER*
- *CINEMO_MENU_POPUP*

With Blu-ray discs this function can simulate seeking to the *CINEMO_TITLE_TOP_MENU* title. In this case, it is however recommended to use [ICinemoPlayer2::SeekTitleChapter\(\)](#) method instead or to post *CINEMO_KEY_POPUP_{ON/OFF}* user key events.

3.20.21 ResumeTitle

Resume title playback (after Stop or ShowMenu).

Syntax

```
CinemoError ResumeTitle();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

With Blu-ray discs the Resume User Operation resumes playback of a previously suspended Movie Object.

3.20.22 ReturnFromSubmenu

Navigate up through menu hierarchy.

Syntax

```
CinemoError ReturnFromSubmenu();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is implemented in accordance with the DVD specifications for “GO_UP” user operations, however it is unlikely to be called by a typical player application.

With Blu-ray discs this feature is not supported.

3.20.23 ReplayChapter

Replay current chapter from beginning.

Syntax

```
CinemoError ReplayChapter();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for tracks which contain multiple chapters, for example DVD-Video discs, VCD/SVCD discs, or audio books.

With Blu-ray discs this feature is not supported.

3.20.24 SetRepeatChapter

Set repeat mode of the chapter.

Syntax

```
CinemoError SetRepeatChapter(  
    [in]      CINEMO_REPEAT CHAPTER repeat  
) ;
```

Parameters

- **repeat**

The chapter repeat mode.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for DVD-Video discs. The repeat mode is set in the scope of the player and valid until changed by a subsequent call of this method. This method does not check for prohibited user operations. A chapter is not repeated if the related user operation is prohibited.

3.20.25 SetRepeatTitle

Set repeat mode of the title.

Syntax

```
CinemoError SetRepeatTitle(  
    [in]      CINEMO_REPEAT_TITLE repeat  
) ;
```

Parameters

- **repeat**

The title repeat mode.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method only succeeds for DVD-Video discs. The repeat mode is set in the scope of the player and valid until changed by a subsequent call of this method. This method does not check for prohibited user operations. A title is not repeated if the related user operation is prohibited.

3.20.26 StepForward

Step forward one frame.

Syntax

```
CinemoError StepForward();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the play speed is not currently zero, this method will have the same effect as *SetSpeed(0)*, to pause playback. Otherwise, playback will advance by a single video frame.

3.20.27 StepBackward

Step backward one frame.

Syntax

```
CinemoError StepBackward();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the play speed is not currently zero, this method will have the same effect as *SetSpeed(0)*, to pause playback. Otherwise, playback will step backward a single video frame.

3.20.28 SelectAudio

This method has the same behavior as [ICinemoPlayer::SelectAudio\(\)](#).

3.20.29 SelectAngle

This method has the same behavior as [ICinemoPlayer::SelectAngle\(\)](#).

3.20.30 SelectSubpicture

This method has the same behavior as [ICinemoPlayer::SelectSubpicture\(\)](#).

3.20.31 AddExternalSubtitle

This method has the same behavior as [ICinemoPlayer::AddExternalSubtitle\(\)](#).

3.20.32 GetStatus

This method has the same behavior as [ICinemoPlayer::GetStatus\(\)](#).

3.20.33 GetGraphStatus

This method has the same behavior as [ICinemoPlayer::GetGraphStatus\(\)](#).

3.20.34 GetQualityInfo

This method has the same behavior as [ICinemoPlayer::GetQualityInfo\(\)](#).

3.20.35GetPosition

This method has the same behavior as [ICinemoPlayer::GetPosition\(\)](#).

3.20.36 GetDuration

This method has the same behavior as [ICinemoPlayer::GetDuration\(\)](#).

3.20.37 GetBuffered

This method has the same behavior as [ICinemoPlayer::GetBuffered\(\)](#).

3.20.38 GetAudio

This method has the same behavior as [ICinemoPlayer::GetAudio\(\)](#).

3.20.39 GetAngle

Get angle or video stream attributes.

Syntax

```
CinemoError GetAngle(  
    [in]  uint32 id,  
    [out] CinemoMediaType& media  
) ;
```

Parameters

- **id**

Stream identifier.

- **media**

Returned video stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The angle or video stream identifier is the one-based index of the video stream in the current track. If the function succeeds, the returned media type will completely describe this stream.

With Blu-ray discs angle and primary video streams are not always mapped one to one. In this case this method is limited for the currently selected angle.

3.20.40 GetSubpicture

This method has the same behavior as [ICinemoPlayer::GetSubpicture\(\)](#).

HUMAX Confidential

3.20.41 GetMediaInfo

This method has the same behavior as [ICinemoPlayer::GetMediaInfo\(\)](#).

HUMAX Confidential

3.20.42 GetTitle

Get information about a title.

Syntax

```
CinemoError GetTitle(
    [in/out] CinemoTitle& info
);
```

Parameters

- **info**

title information structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method takes a *CinemoTitle* structure that will be filled with information about a given title. The *CinemoTitle* structure is defined here:

```
typedef struct {
    uint32 title_id;                                /* title */
    uint32 num_chapters;                            /* chapters */
    uint32 num_audio;                               /* audio streams */
    uint32 num_angle;                               /* angles */
    uint32 num_subpicture;                          /* subpicture streams */
    uint32 num_audio_unsupported;                  /* unsupported audio streams */
    uint32 num_angle_unsupported;                  /* unsupported angles */
    uint32 num_subpicture_unsupported;             /* unsupported subpicture streams */
    uint32 num_subpicture_styles;                  /* subpicture styles (BD nav specific) */
    uint32 num_secondary_video;                   /* secondary video streams (BD nav specific) */
    uint32 num_secondary_audio;                   /* secondary audio streams (BD nav specific) */
    CinemoTitleFlags flags;                        /* title type and permissions (BD nav specific) */
    uint32 reserved;                               /* reserved */
    pts70mhz duration;                            /* 70.56Mhz duration */
} CinemoTitle;
```

You will need to assign a value to the *title_id* field before calling this function to specify the title you want to get information from. The value of this field might be a specific title index, or 0 to get information on the current title.

With Blu-ray discs only the *flags* field is available for all titles after *CINEMO_EC_TITLE* with the current title set to zero and the total number of titles being greater than zero has been received. As soon *CINEMO_EC_TITLE* with a valid current title value was received, the additional information can be queried for that title, but because of the nature of Blu-ray media these attributes can change during playback. Instead of using this method it's recommended to monitor the change events related to the specific attributes.

3.20.43 GetTitleFlags

Get flag only information about a title.

Syntax

```
CinemoError GetTitleFlags(
    [in]  uint32 id,
    [out] CinemoTitleFlags& flags
);
```

Parameters

- **id**
Title identifier.
- **info**
Returned title flags structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The title identifier is the one-based title index. This is a simplified version of [ICinemoPlayer2::GetTitle\(\)](#) method when only the flags are needed example for showing title menu. With some media types querying the full *CinemoTitle* structure would be slow needing additional disc access. The *CinemoTitleFlags* structure is defined here:

```
typedef struct {
    uchar prohibited      : 1; /* title is not selectable (BD nav specific) */
    uchar hidden          : 1; /* title should not be shown in user interface */
    uchar bdj             : 1; /* title is using BD-J (BD nav specific) */
    uchar interactive     : 1; /* title is interactive (BD nav specific) */
    uchar reserved[3];
} CinemoTitleFlags;
```

3.20.44 GetChapter

Get information about a chapter.

Syntax

```
CinemoError GetChapter(  
    [in/out] CinemoChapter& info  
) ;
```

Parameters

- **info**
chapter information structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method takes a *CinemoChapter* structure that will be filled with information about a given chapter. The *CinemoChapter* structure is defined here:

```
typedef struct {  
    uint32 title_id;                      /* title */  
    uint32 chapter_id;                    /* chapter */  
    pts70mhz start;                     /* 70.56Mhz offset */  
    pts70mhz duration;                  /* 70.56Mhz duration */  
} CinemoChapter;
```

You will need to assign a value to the *title_id* and *chapter_id* fields before calling this function to specify the chapter you want to get information from and its related title. The value of these fields might be a specific title and chapter index, or *0* to get information on the current chapter of the current title.

With Blu-ray discs this method will always return *CinemoErrorNotImplemented*.

3.20.45 SetSessionData

This method has the same behavior as [ICinemoPlayer::SetSessionData\(\)](#).

3.20.46 GetSessionPool

This method has the same behavior as [ICinemoPlayer::GetSessionPool\(\)](#).

3.20.47 InitMetapool

This method has the same behavior as [ICinemoPlayer::InitMetapool\(\)](#).

3.20.48 InitPlaylistMetapool

This method has the same behavior as [ICinemoPlayer::InitPlaylistMetapool\(\)](#).

3.20.49 InitPainter

This method has the same behavior as [ICinemoPlayer::InitPainter\(\)](#).

3.20.50 SaveState

Save current playback state.

Syntax

```
CinemoError SaveState (
    [out]    struct ICinemoBlob ** pp
);
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoBlob](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will save the current playback state to a ‘blob’. The state may be subsequently restored by means of the [RestoreState\(\)](#) method.

With Blu-ray discs this feature is not supported because it’s not possible to save the BDJ state. Authoring guidelines recommend that the discs provide a bookmarking functionality themselves.

3.20.51 RestoreState

Restore current playback state.

Syntax

```
CinemoError RestoreState (
    [in]     const void * pdata,
    [in]     uint32 nbytes
);
```

Parameters

- **pdata**

The address of a ‘blob’ of data.

- **nbytes**

The size of the ‘blob’ in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will restore the playback state previously saved by means of the [SaveState\(\)](#) method. It is required that the track for which the state is intended be currently open, so the correct sequence of calls to restore a playback state is:

4. [OpenTrack\(\)](#)
5. Wait for the *CINEMO_EC_OPEN* indicating *CinemoNoError*
6. [RestoreState\(\)](#)

With Blu-ray discs this feature is not supported because it’s not possible to save the BDJ state. Authoring guidelines recommend that the discs provide a bookmarking functionality themselves.

3.20.52 SignalPlaylistChanged

This method has the same behavior as [ICinemoPlayer::SignalPlaylistChanged\(\)](#).

3.20.53 TakeSnapshot

Take a snapshot of the current video frame.

Syntax

```
CinemoError TakeSnapshot(  
    [in]     CinemoThumbFormat& thumb          /* format for snapshot */  
) ;
```

Parameters

- **thumb**

The format for the requested snapshot.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will take a snapshot of the current video frame, encoded with the specified format and scaled to not more than the specified size in pixels. If successful, the result is stored in the metapool under the metaname *CINEMO_METANAME_SNAPS*, and the *CINEMO_EC_METADATA* event is posted.

With content protected Blu-ray discs this function always returns a *CinemoError* code.

3.20.54 GetWindowDeviceName

Get window device name string with platform specific window or surface handles as URL parameters.

Syntax

```
CinemoError GetWindowDeviceName (
    [out]    struct ICinemoUTF8 ** pp
);
```

Parameters

- **pp**

The returned device name value.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.20.55 SelectSubpictureStyle

Select a style for current subpicture stream.

Syntax

```
CinemoError SelectSubpictureStyle(  
    [in]      uint32 id  
) ;
```

Parameters

- **id**

Subpicture style identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The subpicture style identifier is one-based index of the subpicture style for the current subpicture stream.

3.20.56 SelectSecondaryAudio

Select a secondary audio stream.

Syntax

```
CinemoError SelectSecondaryAudio(  
    [in]      uint32 id  
) ;
```

Parameters

- **id**

Secondary audio stream identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The secondary audio stream identifier may be either zero, or the one-based index of the secondary audio stream in the current track. If set to zero, secondary audio output will be disabled (i.e. no secondary audio stream will be selected).

3.20.57 SelectSecondaryVideo

Select a secondary video stream.

Syntax

```
CinemoError SelectSecondaryVideo(  
    [in]      uint32 id  
) ;
```

Parameters

- **id**

Secondary video stream identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The secondary video stream identifier may be either zero, or the one-based index of the secondary video stream in the current track. If set to zero, secondary video output will be disabled (i.e. no secondary video stream will be selected).

3.20.58 GetSecondaryAudio

Get secondary audio stream attributes.

Syntax

```
CinemoError GetSecondaryAudio (
    [in]  uint32 id,
    [out] CinemoMediaType& media
);
```

Parameters

- **id**
Stream identifier.
- **media**
For returning secondary audio stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The secondary audio stream identifier is the one-based index of the secondary audio stream in the current track.
If the function succeeds, the returned media type will completely describe this stream.

3.20.59 GetSecondaryVideo

Get secondary video stream attributes.

Syntax

```
CinemoError GetSecondaryVideo (
    [in]  uint32 id,
    [out] CinemoMediaType& media
);
```

Parameters

- **id**
Stream identifier.
- **media**
For returning secondary video stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The secondary video stream identifier is the one-based index of the secondary video stream in the current track.
If the function succeeds, the returned media type will completely describe this stream.

3.20.60 PostKeyUserEvent

Post user key event to player.

Syntax

```
CinemoError PostKeyUserEvent (
    [in]  CINEMO_KEYEVENT event,
    [in]  CINEMO_KEY code,
    [in]  sint32 arg
);
```

Parameters

- **event**

Press or release type of key event.

- **code**

User key event identifier.

- **arg**

Optional argument to key event.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Returning from this function doesn't guarantee the key event was already processed or the action is executed. It's possible the action is masked by the content application.

According to the Blu-ray specification "key events have no defined semantic meaning". Of course content providers are encouraged to bind the functionality of the key events to an intuitive meaning for the user. Since content providers are free in their decision how to react on certain key presses from the user, it is not recommended that the HMI assumes certain states of playback according to the generated key event.

The *CINEMO_KEYEVENT* does not have the "CLICKED" key-event. In case the underlying protocol supports only a "CLICKED" key-event the user application can post *CINEMO_KEYEVENT_RELEASED* immediately after posting *CINEMO_KEYEVENT_PRESSED*. Furthermore, as convention the most components in Cinemo SDK perform actions upon *CINEMO_KEYEVENT_RELEASED* in that case. Usually *CINEMO_KEYEVENT_RELEASED* shall be posted after the corresponding *CINEMO_KEY_PRESSED*, however, the timespan between the two events depends upon user-interaction and behavior of user application.

Cinemo defines a generic set of key user events. These names of the Cinemo defined key user events not always map to the action name on the target device. Therefore the following table give you an overview of what key user events are supported for the different playback scenarios. Keys not supported will return with *CinemoErrorImpossible*.

Later on you will find tables mapping the Cinemo key names to the actual functions performed on the target device.

KEY	MSD	DVD	Blu-ray	iAP	AVRCP	Cloud
CINEMO_KEY_PLAY	✓	✓	✓	✓	✓	✓
CINEMO_KEY_STOP_TITLE	✓	✓	✓			✓
CINEMO_KEY_PAUSE_ON	✓	✓	✓	✓	✓	✓
CINEMO_KEY_PAUSE_OFF	✓	✓	✓	✓		✓
CINEMO_KEY_STILL_OFF			✓			
CINEMO_KEY_TRICK_PLAY	✓	✓	✓		✓	✓
CINEMO_KEY_NEXT CHAPTER	✓	✓	✓	✓	✓	✓
CINEMO_KEY_PREV CHAPTER	✓	✓	✓	✓	✓	✓
CINEMO_KEY_SECONDARY_VIDEO_ON			✓			
CINEMO_KEY_SECONDARY_VIDEO_OFF			✓			
CINEMO_KEY_SECONDARY_AUDIO_ON			✓			
CINEMO_KEY_SECONDARY_AUDIO_OFF			✓			
CINEMO_KEY_SUBPICTURE_ON	✓	✓	✓			✓
CINEMO_KEY_SUBPICTURE_OFF	✓	✓	✓			✓
CINEMO_KEY_INTERACTIVE_LEFT		✓	✓			
CINEMO_KEY_INTERACTIVE_UP		✓	✓			
CINEMO_KEY_INTERACTIVE_RIGHT		✓	✓			
CINEMO_KEY_INTERACTIVE_DOWN		✓	✓			
CINEMO_KEY_INTERACTIVE_ENTER		✓	✓			
CINEMO_KEY_INTERACTIVE_POPUP_OFF			✓			
CINEMO_KEY_INTERACTIVE_POPUP_ON			✓			
CINEMO_KEY_INTERACTIVE_SELECT		✓	✓			
CINEMO_KEY_INTERACTIVE_SELECT_AND_ACTIVATE	✓	✓				
CINEMO_KEY_NUMERIC_0..9			✓			
CINEMO_KEY_COLOURED_0			✓			
CINEMO_KEY_COLOURED_1			✓			
CINEMO_KEY_COLOURED_2			✓			
CINEMO_KEY_COLOURED_3			✓			
CINEMO_KEY_PLAYLIST_REPEAT	✓			✓		✓
CINEMO_KEY_PLAYLIST_ORDER	✓			✓		✓
CINEMO_KEY_REMOTE_VOLUME				✓	✓	
CINEMO_KEY_REMOTE_VOLUME_UP				✓	✓	
CINEMO_KEY_REMOTE_VOLUME_DOWN				✓	✓	

KEY	MSD	DVD	Blu-ray	iAP	AVRCP	Cloud
CINEMO_KEY_PROMOTE				✓		✓
CINEMO_KEY_DEMOTE				✓		✓
CINEMO_KEY_ADD_TO_CART				✓		

Mapping of Cinemo keys to MSD playback

This table shows the mapping of the key events to playback from mass storage devices (MSD). This is also a replacement of functions missing from [ICinemoPlayer2](#) to functions present in [ICinemoPlayer](#) implementation.

KEY	MSD playback function	triggered at
CINEMO_KEY_PLAY	ICinemoPlayer::Play() without ICinemoPlayer::SetSpeed(1000)	release
CINEMO_KEY_STOP_TITLE	ICinemoPlayer::Stop()	release
CINEMO_KEY_PAUSE_ON	ICinemoPlayer::SetSpeed(0)	release
CINEMO_KEY_PAUSE_OFF	ICinemoPlayer::SetSpeed(1000)	release
CINEMO_KEY_TRICK_PLAY	ICinemoPlayer::SetSpeed(arg)	release
CINEMO_KEY_NEXT CHAPTER	ICinemoPlayer::NextChapter(0)	release
CINEMO_KEY_PREV CHAPTER	ICinemoPlayer::PreviousChapter(0)	release
CINEMO_KEY_SUBPICTURE_ON	ICinemoPlayer::SelectSubpicture(1)	release
CINEMO_KEY_SUBPICTURE_OFF	ICinemoPlayer::SelectSubpicture(0)	release
CINEMO_KEY_PLAYLIST_REPEAT	ICinemoPlaylist::SetRepeat() cycle through Off/All/Single	release
CINEMO_KEY_PLAYLIST_ORDER	ICinemoPlaylist::SetOrder() cycle through Sequential/Shuffle	release

The term “release” in the “triggered at” column means the MSD playback function is executed when the [ICinemoPlayer2::PostUserKeyEvent\(\)](#) has been called with *CINEMO_KEYEVENT_RELEASED* in the event parameter.

Mapping of Cinemo keys to DVD playback

This table shows the mapping of the key events to playback DVD formats. It shows the equivalent function of [ICinemoPlayer](#) which is not present in [ICinemoPlayer2](#).

KEY	DVD playback function	triggered at
CINEMO_KEY_PLAY	ICinemoPlayer::Play() without ICinemoPlayer::SetSpeed(1000)	release
CINEMO_KEY_STOP_TITLE	ICinemoPlayer::Stop()	release
CINEMO_KEY_PAUSE_ON	ICinemoPlayer::SetSpeed(0)	release

KEY	DVD playback function	triggered at
CINEMO_KEY_PAUSE_OFF	ICinemoPlayer::SetSpeed(1000)	release
CINEMO_KEY_TRICK_PLAY	ICinemoPlayer::SetSpeed(arg)	release
CINEMO_KEY_NEXT CHAPTER	ICinemoPlayer::NextChapter(0)	release
CINEMO_KEY_PREV CHAPTER	ICinemoPlayer::PreviousChapter(0)	release
CINEMO_KEY_SUBPICTURE_ON	ICinemoPlayer::SelectSubpicture(1)	release
CINEMO_KEY_SUBPICTURE_OFF	ICinemoPlayer::SelectSubpicture(0)	release
CINEMO_KEY_INTERACTIVE_LEFT	Left arrow key	release
CINEMO_KEY_INTERACTIVE_UP	Up arrow key	release
CINEMO_KEY_INTERACTIVE_RIGHT	Right arrow key	release
CINEMO_KEY_INTERACTIVE_DOWN	Down array key	release
CINEMO_KEY_INTERACTIVE_ENTER	Enter (OK) key	release
CINEMO_KEY_INTERACTIVE_SELECT	Select button. Argument is the button identifier.	release
CINEMO_KEY_INTERACTIVE_SELECT_AND_ACTIVATE	Select and activate button. Argument is the button identifier.	release

The term “release” in the “triggered at” column means the DVD playback function is executed when the [ICinemoPlayer2::PostUserKeyEvent\(\)](#) has been called with *CINEMO_KEYEVENT_RELEASED* in the event parameter.

Mapping of Cinemo keys to Blu-ray playback

This table shows which keys are mandatory or optional (according to Blu-ray specification) and the corresponding mapping to Blu-ray key events.

KEY	Blu-ray M/O	DESCRIPTION
CINEMO_KEY_PLAY	mandatory	Play button
CINEMO_KEY_STOP_TITLE	mandatory	Stop button
CINEMO_KEY_PAUSE_ON	optional	Pause playback ON
CINEMO_KEY_PAUSE_OFF	optional	Pause playback OFF
CINEMO_KEY_STILL_OFF	mandatory	Still OFF
CINEMO_KEY_TRICK_PLAY	optional	Fast forward or fast backward depending on argument speed
CINEMO_KEY_NEXT CHAPTER	mandatory	Go to next chapter
CINEMO_KEY_PREV CHAPTER	mandatory	Go to previous chapter
CINEMO_KEY_SECONDARY_VIDEO_ON	optional	Enabling secondary video

KEY	Blu-ray M/O	DESCRIPTION
CINEMO_KEY_SECONDARY_VIDEO_OFF	optional	Disabling secondary video
CINEMO_KEY_SECONDARY_AUDIO_ON	optional	Enabling secondary audio
CINEMO_KEY_SECONDARY_AUDIO_OFF	optional	Disabling secondary audio
CINEMO_KEY_SUBPICTURE_ON	mandatory	Enabling subpicture
CINEMO_KEY_SUBPICTURE_OFF	mandatory	Disabling subpicture
CINEMO_KEY_INTERACTIVE_LEFT	mandatory	Left arrow key
CINEMO_KEY_INTERACTIVE_UP	mandatory	Up arrow key
CINEMO_KEY_INTERACTIVE_RIGHT	mandatory	Right arrow key
CINEMO_KEY_INTERACTIVE_DOWN	mandatory	Down arrow key
CINEMO_KEY_INTERACTIVE_ENTER	mandatory	Enter (OK) key
CINEMO_KEY_INTERACTIVE_POPUP_OFF	mandatory	Popup menu OFF
CINEMO_KEY_INTERACTIVE_POPUP_ON	mandatory	Popup menu ON
CINEMO_KEY_INTERACTIVE_SELECT	optional	Select button. Argument is the button identifier.
CINEMO_KEY_INTERACTIVE_SELECT_AND_ACTIVATE	optional	Select and activate button. Argument is the button identifier.
CINEMO_KEY_NUMERIC_0..9	mandatory	Numeric keys from 0 to 9
CINEMO_KEY_COLOURED_0	mandatory	Left most colored key
CINEMO_KEY_COLOURED_1	mandatory	Second left most colored key
CINEMO_KEY_COLOURED_2	mandatory	Second right most colored key
CINEMO_KEY_COLOURED_3	mandatory	Right most colored key

Mapping of Cinemo keys to iAP functions on Apple devices

This table shows the mapping of the Cinemo key name to the corresponding iAP key function as defined in Apples MFi specifications.

KEY	Apple device function
CINEMO_KEY_PLAY	Play
CINEMO_KEY_PAUSE_ON	Pause
CINEMO_KEY_PAUSE_OFF	Play
CINEMO_KEY_NEXT CHAPTER	Next (holding key pressed scrubs forward until released)
CINEMO_KEY_PREV CHAPTER	Previous (holding key pressed scrubs backward until released)
CINEMO_KEY_PLAYLIST_REPEAT	Repeat
CINEMO_KEY_PLAYLIST_ORDER	Shuffle
CINEMO_KEY_REMOTE_VOLUME	Mute (arg must be 0)
CINEMO_KEY_REMOTE_VOLUME_UP	Louder

KEY	Apple device function
CINEMO_KEY_REMOTE_VOLUME_DOWN	Softer
CINEMO_KEY_PROMOTE	Play More Like This
CINEMO_KEY_DEMOTE	Play Less Like This
CINEMO_KEY_ADD_TO_CART	Add to Wish List

Mapping of Cinemo keys to functions on devices supporting AVRCP

This table shows the mapping of the Cinemo key name to the corresponding function as defined by the AVRCP specification

KEY	AVRCP function
CINEMO_KEY_PLAY	Play
CINEMO_KEY_PAUSE_ON	Pause
CINEMO_KEY_REMOTE_VOLUME	Set volume
CINEMO_KEY_REMOTE_VOLUME_UP	Louder
CINEMO_KEY_REMOTE_VOLUME_DOWN	Softer

Mapping of Cinemo keys to Cloud playback

This table shows the mapping of the key events to playback from Cinemo Distributed Cloud services. Cloud playback essentially has the same features as MSD playback, depending on the underlying format and cloud service.

The following table contains the keys that may be used to interact with cloud service tracks in addition to the standard MSD keys.

KEY	Cloud playback function
CINEMO_KEY_PROMOTE	Like / Promote (Like) currently played content
CINEMO_KEY_DEMOTE	Dislike / Demote (Dislike) currently played content

3.20.61 PostKeyEvent

Post generic key event to player.

Syntax

```
CinemoError PostKeyEvent(  
    [in]  CINEMO_KEYEVENT event,  
    [in]  CINEMO_KEYMODIFIER modifier,  
    [in]  uint32 code  
) ;
```

Parameters

- **event**

Press or release type of key event.

- **modifier**

Type of key modifier or normal key code.

- **code**

Platform dependent key code.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Returning from this function doesn't guarantee the key event was already processed.

For normal keys the modifier should be *CINEMO_KEYMODIFIER_NONE*. Pressing or releasing modifier keys should be sent separately by using the modifier parameter and passing zero as code.

3.20.62 PostMouseEvent

Post mouse event to player.

Syntax

```
CinemoError PostMouseEvent (
    [in] CINEMO_MOUSEEVENT event,
    [in] const CinemoPoint& point
);
```

Parameters

- **event**

Type of mouse event.

- **point**

Pixel coordinates of the mouse event.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Returning from this function doesn't guarantee the mouse event was already processed.

Since mouse movement and clicks are optional actions, almost all commercial Blu-ray discs with BD-J do not react on mouse interactions. They do react however on the mandatory set of key actions and operations.

3.20.63 GetTrackURL

Get URL for the currently playing track from the playlist or track provider.

Syntax

```
CinemoError GetTrackURL(  
    [out]    struct ICinemoUTF8 ** pp  
) ;
```

Parameters

- **pp**

The returned URL.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.20.64 GetTrackVFSAttributes

Get VFS attributes for the currently playing track.

Syntax

```
CinemoError GetTrackVFSAttributes (
    [out]    CinemoVFSAttributes& attrs
);
```

Parameters

- **attrs**

The returned attributes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.20.65 GetDistributedServerURL

Get the URL of the Cinemo Distributed Playback Server.

Syntax

```
CinemoError GetDistributedServerURL(  
    [out]    struct ICinemoUTF8 ** pp  
);
```

Parameters

- **pp**

The returned URL.

Return value

If the method succeeds, it returns *CinemoNoError*. It returns *CinemoFalse* if the distributed server is disabled. Otherwise, it returns a *CinemoError* code.

Remarks

This method returns the URL of the distributed server.

3.21 ICinemoControlPoint

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

The [ICinemoControlPoint](#) interface implements a UPnP Control Point. It can be created using the `CinemoCreateControlPoint()` function defined in `cinemo_mm.h`. Please note that this interface is not thread-safe.

The [ICinemoControlPoint](#) derives from the [ICinemoPlayer](#) interface. Note that not all functions of the [ICinemoPlayer](#) are supported to be called on a control point. The exact set of functions supported depends on the media renderer type and feature set. i.e. when connected to a Cinemo Media Renderer, the majority of the [ICinemoPlayer](#) functions are available, while if connected to a generic uPnP Media Renderer, the set of functions supported is restricted to what's defined on the UPnP specification and actually supported by the media renderer. Functions on the derived [ICinemoPlayer](#) interface that are not supported by the media renderer return `CinemoErrorNotImplemented`.

A Cinemo Media Renderer can be created using the `CinemoCreateMediaRenderer()` function defined in `cinemo.h`.

ICinemoControlPoint Methods

METHOD	DESCRIPTION
ConnectTo	Connect to an existing media renderer on the network
Cancel	Cancel all operations
Enable	Enable all operations after a call to Cancel
GetRendererType	Get the type of the media renderer
GetRendererCaps	Get the capabilities of the media renderer
IsPlayable	Checks if a track on the playlist is playable on the media renderer
GetPlaylist	Get the ICinemoPlaylist instance that is attached to this control point

3.21.1 ConnectTo

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Connect to an existing media renderer on the network.

Syntax

```
CinemoError ConnectTo(
    [in] const char * szurl,
    [in] uint32 flags);
```

Parameters

- **szurl**
The URL of the media renderer.
- **flags**
CINEMO_CONTROLPOINT_FLAGS bitmask.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The URL of the media renderer, if not explicitly known, can be obtained by calling [ICinemoVFS::Open\(\)](#) in UPnP discovery mode (see section 0). It is returned as the VFS_PATH attribute of discovered media renderers. The following is an example URL for a Cinemo Media Renderer running locally on port number 49158:

```
upnp://127.0.0.1:49158/0/MediaRenderer/DeviceDesc.xml
```

The *flags* argument specifies the behavior of the control point and the [ConnectTo\(\)](#) method:

- **CINEMO_CONTROLPOINT_FLAGS_NONE**
Placeholder that can be used if none of the flags below is set. Equals 0.
- **CINEMO_CONTROLPOINT_FLAGS_READ**
If this flag is set, the method will block until it is finished and return the corresponding error code. the [Cancel\(\)](#) method can be used to abort this blocking call. If this flag is not set, the method returns *CinemoErrorPending* immediately and the application has to wait for the *CINEMO_EC_OPEN* event with track id 0.
- **CINEMO_CONTROLPOINT_FLAGS_UPNP_ONLY**
If this flag is set, then the connection to the renderer will be done using only UPnP services, even if the renderer supports distributed playback. Otherwise a distributed playback connection will be preferred.

3.21.2 Cancel

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

3.21.3 Enable

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Cancel or enable [ICinemoControlPoint](#) and derived [ICinemoPlayer](#) methods.

Syntax

```
CinemoError Cancel();  
CinemoError Enable();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

[Cancel\(\)](#) will unblock any call to all [ICinemoControlPoint](#) and derived [ICinemoPlayer](#) methods. Like all other Cinemo API's, the [Cancel\(\)](#) operation is sticky – after calling [Cancel\(\)](#), the [ICinemoControlPoint](#) object will remain in a cancelled state until [Enable\(\)](#) is called. This is to avoid potential race conditions.

If a method is unblocked by [Cancel\(\)](#), it will return *CinemoErrorCancel* error code.

Since all [ICinemoControlPoint](#) and derived [ICinemoPlayer](#) methods involve connecting by HTTP or otherwise to a local or remote Media Renderer, depending on network conditions, they may potentially block for a long time, or infinitely. Correct use of [ICinemoControlPoint](#) methods requires the calling application to maintain threads and implement synchronization strategies to avoid hang ups.

3.21.4 GetRendererType

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Get the type of the media renderer, which the control point is connected to.

Syntax

```
CINEMO_RENDERER_TYPE GetRendererType()
```

Return value

The method returns CINEMO_RENDERER_TYPE_CINEMO if the media renderer is a Cinemo Media Renderer. It returns CINEMO_RENDERER_TYPE_GENERIC if the media renderer is a generic uPnP Media Renderer, or CINEMO_RENDERER_TYPE_UNKNOWN if the [ICinemoControlPoint](#) instance is not connected to any renderer.

Remarks

Depending on the type of media renderer, a different set of functions is supported on the derived [ICinemoPlayer](#) interface. i.e. when connected to a Cinemo Media Renderer, the majority of the [ICinemoPlayer](#) functions are available, while if connected to a generic UPnP Media Renderer, the set of functions supported is restricted to what's defined on the UPnP specification and actually supported by the media renderer.

Functions on the derived [ICinemoPlayer](#) interface that are not supported by the media renderer return *CinemoErrorNotImplemented*.

3.21.5 GetRendererCaps

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Get the capabilities of the media renderer, which the control point is connected to.

Syntax

```
CINEMO_RENDERER_CAPS GetRendererCaps()
```

Return value

The method returns a set of bit-flags that indicate the capabilities of the media renderer.

Remarks

This method can return a combination of the following flags:

- **CINEMO_RENDERER_CAPS_NONE**
No special capabilities.
- **CINEMO_RENDERER_CAPS_GAPLESS**

The media renderer supports gap-less next track advance. Note that a media renderer without support for gap-less next track will not automatically continue to play the next track on the Control Point playlist after reaching end of a track, but the playback needs to be triggered manually via PlayTrack call on the [ICinemoControlPoint](#)

3.21.6 IsPlayable

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Checks if a track on the playlist is playable on the media renderer.

Syntax

```
CinemoError IsPlayable(uint64 track_id)
```

Return value

If the track can played on the media renderer, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

A media renderer might not support the same set of media formats as supported by the local Cinemo Media Engine. *CINEMO_VFS_FLAG_PLAYABLE* as returned by the [ICinemoPlaylist](#) refers to the local Cinemo Media Engine. Use this function instead to check if a track is playable on the remote media renderer.

3.21.7 GetPlaylist

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

Get the [ICinemoPlaylist](#) instance that is attached to this control point.

Syntax

```
CinemoError GetPlaylist(  
    [out] struct ICinemoPlaylist** pp)
```

Parameters

- **pp**

The attached playlist instance.

Return value

If the method succeeds, it returns CinemoNoError. Otherwise, it returns a CinemoError code.

Remarks

The function returns the [ICinemoPlaylist](#) instance as passed to [CinemoCreateControlPoint](#).

3.22 ICinemoTrackCopier

The [ICinemoTrackCopier](#) interface provides thread-safe methods for copying tracks. It can be obtained by calling [ICinemoUnknown::QueryInterface\(\)](#) on an existing [ICinemoPlaylist](#) instance.

ICinemoTrackCopier Methods

METHOD	DESCRIPTION
SetCopierCallback	Set callback function for receiving copier events
SetCopierSourceID	Set source ID for posted track copier events
SetCopierTarget	Set target folder and encode format
SetCopierTrackTarget	Set custom target file for specific playlist entry
SetCopierTrackFormat	Set custom encode format for specific playlist entry
SetCopierTrackStatus	Restore saved status for specific playlist entry
SetCopierTimeEventsInterval	Set interval between copier progress events
GetCopierStatus	Get status of the copier
GetCopierTrackStatus	Get status of an individual track
EnableCopier	Enable track copier
DisableCopier	Disable track copier
ResetCopier	Reset track copier

Notes

[ICinemoPlaylist](#) instances which expose [ICinemoTrackCopier](#) interfaces are capable of:

1. Copying tracks
2. Playing tracks
3. Copying and playing tracks at the same time.

This behaviour, with certain limitations, is designed to be transparent to the calling application. Cinemo will avoid the copying and playback of **different** tracks at the **same** time, which could cause extensive thrashing of, for example, a CD device making playback impossible. To achieve this goal, when playback of a track is requested,

- If the source track has already been copied, or is currently copying, then its target file is played instead (a track may be copied and played at the same time).
- If the source track is queued for copying, and another track is currently copying, then the currently copying track is cancelled, and the source track is copied instead. The cancelled track will remain queued for copying later.

This means, that after calling [EnableCopier\(\)](#), an application should not be concerned with the order in which tracks are copied – tracks will normally be copied sequentially and in order, starting with the first track in the playlist, but this order may be changed if an attached [ICinemoPlayer](#) instance chooses to play tracks in a different order. The application should **not** assume that single tracks are copied sequentially – in future versions

of Cinemo, depending on the source playlist, the interface is designed to support copying of multiple tracks in parallel (e.g. HTTP download).

The application is informed of all [ICinemoTrackCopier](#) state changes by means of an event callback.

If the metadata associated with the [ICinemoPlaylist](#) instance that this [ICinemoTrackCopier](#) was obtained from has been changed before calling [EnableCopier\(\)](#), these changes might be reflected in the output if the selected container format supports it.

3.22.1 SetCopierCallback

Set callback function for receiving copier events.

Syntax

```
CinemoError SetCopierCallback (
    [in]     void * puser,
    [in]     CinemoEventCallback pcall
);
```

Parameters

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The user defined application callback function will be called asynchronously whenever a new copier event is available. The application is responsible for synchronizing events from multiple threads to its own application thread ([ICinemoEventQueue](#) interface may be used for this purpose).

Currently, only the following events will be posted:

- **CINEMO_EC_TRACK**

Indicates the CinemoCopierTrackStatus for the specified track has changed. The 64-bit track identifier is encoded in the d[0] and d[1] fields of the event.

- **CINEMO_EC_FINISHED**

Indicates that copying of all tracks has completed.

To stop receiving events, a NULL callback function may be specified.

3.22.2 SetCopierSourceID

Set source ID for posted track copier events.

Syntax

```
CinemoError SetCopierSourceID(  
    [in]      uint32 source_id  
) ;
```

Parameters

- **source_id**

User-defined number passed in *CinemoEvent* *source_id* member.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This approach helps differentiate sources of events when using shared event callback function.

3.22.3 SetCopierTarget

Set target folder and encode format.

Syntax

```
CinemoError SetCopierTarget(  
    [in]     const char * szfolder,  
    [in]     CINEMO_COPYFORMAT copyformat,  
    [in]     uint32 bitrate  
) ;
```

Parameters

- **szfolder**

The UTF-8 full path name of the target folder. The target folder must exist.

- **copyformat**

The format in which to encode target files.

- **bitrate**

The bit-rate at which to encode target files (zero for default value).

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [SetCopierTarget\(\)](#) method should be called before copying is enabled. It is used to specify the target folder and encoding parameters. The target folder must exist. Tracks will be written to the target folder with an automatically generated filename (e.g. "Track_01.wav"), containing the order of the track in the playlist, and with an appropriate file extension for the target format. Any files with the same file name in the target folder will be overwritten when the track is copied.

3.22.4 SetCopierTrackTarget

Set custom target file for specific playlist entry.

Syntax

```
CinemoError SetCopierTrackTarget(  
    [in]      uint64 track_id,  
    [in]      ICinemoFile * pfile  
) ;
```

Parameters

- **track_id**

The track identifier for which to set to custom target file.

- **pfile**

The custom target file.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [SetCopierTrackTarget\(\)](#) method may only be called before copying is enabled. It is used to specify a custom target file for the given track. Although [ICinemoFile](#) objects created by [CinemoCreateFile\(\)](#) are supported, the recommended usecase for this method is that an object of a custom [ICinemoFile](#) implementation is passed in order to control the actual stream output process. If a custom target file for a specific playlist entry is set, the output file that would normally be generated if [SetCopierTrackTarget\(\)](#) would not be used is not created.

3.22.5 SetCopierTrackFormat

Set custom encode format for specific playlist entry.

Syntax

```
CinemoError SetCopierTrackFormat(  
    [in]     uint64 track_id,  
    [in]     CINEMO_MUXFORMAT container,  
    [in]     const CinemoMediaType& media,  
    [in]     uint32 bitrate  
) ;
```

Parameters

- **track_id**

The track identifier for which to set to custom encode format.

- **container**

The container format to be used for multiplexing the encoded stream.

- **media**

The target media type for the encoding.

- **bitrate**

The bit-rate at which to encode target file (zero for default value).

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [SetCopierTrackFormat\(\)](#) method may only be called before copying is enabled. It is used to specify a custom encode format (rather than the format specified with [SetCopierTarget\(\)](#)) for the given track. While many fields of the *CinemoMediaType* struct may be left blank, it is recommended that at least the type and subtype members of the *CinemoMediaType* are set (e.g. *CINEMO_MEDIATYPE_AUDIO* and *CINEMO_MEDIASUBTYPE_PCM* for PCM output). It is possible to specify optional resampling or remixing if the respective fields in the *CinemoAudioFormat*-part of the media type are set. Media types which specify a media type header are not supported.

3.22.6 SetCopierTrackStatus

Restore saved status for specific playlist entry.

Syntax

```
CinemoError SetCopierTrackStatus(
    [in]      uint64 track_id,
    [in]      const CinemoCopierTrackStatus& status
);
```

Parameters

- **track_id**

The track identifier for which to restore the saved status.

- **status**

The saved status.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [SetCopierTrackStatus\(\)](#) method may only be called before copying is enabled. It is used to restore the saved status from a previous copying run. Only *CINEMO_COPYSTATUS_FINISHED* state entries will be skipped during following copying. Restoring *ppath* is optional, in case of NULL value the target filename will be regenerated.

3.22.7 SetCopierTimeEventsInterval

Set interval between copier progress events.

Syntax

```
CinemoError SetCopierTimeEventsInterval(  
    [in]      uint32 time_events_ms  
) ;
```

Parameters

- **time_events_ms**

The interval, in milliseconds, between *CINEMO_EC_TRACK* events.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

When a track is copying, *CINEMO_EC_TRACK* events are posted at regular intervals to indicate the copy progress. This method may be used to specify the frequency of these events. The default interval is 1000ms.

The [SetCopierTimeEventsInterval\(\)](#) method may be called at any time to dynamically change the interval between *CINEMO_EC_TRACK* events.

3.22.8 GetCopierStatus

Get status of the copier.

Syntax

```
CinemoError GetCopierStatus(
    [out]    CinemoCopierStatus& status
);
```

Parameters

- **status**

For returning the current copier status.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a *CinemoCopierStatus* structure will be initialized with the current copier status. The *CinemoCopierStatus* structure is defined here:

```
typedef struct {
    uint32 enabled;
    CINEMO_COPYFORMAT format;
    uint32 bitrate;
} CinemoCopierStatus;
```

- **enabled**

Boolean value indicates if the track copier is enabled or disabled.

- **format**

The copyformat specified in last call to [SetCopierTarget\(\)](#).

- **bitrate**

The bitrate specified in last call to [SetCopierTarget\(\)](#).

The [GetCopierStatus\(\)](#) method returns information about the state of the copier. To get the copy status of an individual track, use the [GetCopierTrackStatus\(\)](#) method.

3.22.9 GetCopierTrackStatus

Get status of an individual track.

Syntax

```
CinemoError GetCopierTrackStatus(
    [in]     uint64 track_id,
    [out]    CinemoCopierTrackStatus& status
);
```

Parameters

- **track_id**

The track identifier.

- **status**

For returning the current track copier status.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, a *CinemoCopierTrackStatus* structure will be initialized with the current status of the specified track:

```
typedef struct {
    CINEMO_COPYSTATUS state;
    CinemoError error;
    CINEMO_SEEKUNIT units;
    struct ICinemoUTF8 * ppath;
    uint64 position;
    uint64 duration;
} CinemoCopierTrackStatus;
```

- **state**

The current state of the track (e.g. *queued*, *copying*, *finished*).

- **error**

Indicates if an error occurred.

- **ppath**

If set to a valid *ICinemoUTF8* interface before calling [GetCopierTrackStatus\(\)](#), the string will be initialized to the target filename of the track. Otherwise, *ppath* should be NULL.

- **units, position, duration**

The position and duration of the track, and the units of these values.

The state field of the *CinemoCopierTrackStatus* structure indicates the track copy status, and is an enumerated type as follows:

```
typedef enum {
    CINEMO_COPYSTATUS_NONE = 0,
    CINEMO_COPYSTATUS_QUEUED,
    CINEMO_COPYSTATUS_OPEN,
    CINEMO_COPYSTATUS_COPY,
    CINEMO_COPYSTATUS_FINISHED,
} CINEMO_COPYSTATUS;
```

- **CINEMO_COPYSTATUS_NONE**

The track has no state, which can occur if [EnableCopier\(\)](#) has not yet been called, or if the track was appended to the playlist after calling [EnableCopier\(\)](#).

- **CINEMO_COPYSTATUS_QUEUED**

The track is queued for copying.

- **CINEMO_COPYSTATUS_OPEN**

The track is currently being opened, with the intention to copy.

- **CINEMO_COPYSTATUS_COPY**

The track is currently copying.

- **CINEMO_COPYSTATUS_FINISHED**

The track has finished copying, the error code indicates if copying succeeded or not.

3.22.10 EnableCopier

Enable track copier.

Syntax

```
CinemoError EnableCopier();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [EnableCopier\(\)](#) method will create the track copier thread and start the copying of any queued tracks. The status of the track copier can be monitored by means of posted events.

3.22.11 DisableCopier

Disable track copier.

Syntax

```
CinemoError DisableCopier();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [DisableCopier\(\)](#) method will terminate the track copier thread. Any tracks which are currently being copied will be aborted, and will enter *CINEMO_COPYSTATUS_QUEUED* state.

After calling [DisableCopier\(\)](#), it is possible to again call [EnableCopier\(\)](#), in which case only the tracks queued for copying will be copied – all tracks which finished copying before calling [DisableCopier\(\)](#) will not be copied again, and will remain in *CINEMO_COPYSTATUS_FINISHED* state.

3.22.12 ResetCopier

Reset track copier.

Syntax

```
CinemoError ResetCopier();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [ResetCopier\(\)](#) method will terminate the track copier thread. Any tracks which are currently being copied will be aborted. All tracks will enter *CINEMO_COPYSTATUS_QUEUED* state.

After calling [ResetCopier\(\)](#), it is possible to again call [EnableCopier\(\)](#), in which case **all** tracks in the source playlist will be queued for copying. [ResetCopier\(\)](#) can be used to copy a playlist multiple times to different copier targets.

3.23 ICinemoTrackProvider

The [ICinemoTrackProvider](#) interface provides low level thread-safe playlist functions for use of [ICinemoPlayer](#) in *standalone mode*, that is, without using any Cinemo supplied [ICinemoPlaylist](#) objects. This interface must be implemented by the calling application.

To avoid unnecessary development overhead, this low-level interface is expected to be used only as a last resort option, i.e. when the functionality already exposed by Cinemo supplied playlists is insufficient or unsuitable for your application, and after consultation with Cinemo.

ICinemoTrackProvider Methods

METHOD	DESCRIPTION
AttachPlayer	Attach a player instance
DetachPlayer	Detach a player instance
IsValidTrack	Check if a track is valid
GetNextTrack	Get the next track
GetPrevTrack	Get the previous track
GetTrackInfo	Get the URL or additional metadata for a track
GetRepeat	Get repeat mode
GetOrder	Get play order
SetRepeatAsync	Set repeat mode of the playlist
SetOrderAsync	Set play order of the playlist

Notes

Dynamic playlists – that is, playlists which may change during playback – are fully supported by the methods in this interface. If the tracks in a playlist change, [ICinemoPlayer::SignalPlaylistChanged\(\)](#) should be called on all attached players. This will cause the players to re-evaluate both their currently playing track and any next or previous track which may be opened and queued for gapless playback.

For example, after calling [ICinemoPlayer::SignalPlaylistChanged\(\)](#), if the currently playing track is found to be invalid because [IsValidTrack\(\)](#) returns an error then the player will stop playback of the track and attempt to advance by calling [GetNextTrack\(\)](#). In this case, the invalid track identifier will still be passed to [GetNextTrack\(\)](#) so the implementation of this interface must correctly handle this scenario. Cinemo playlists internally handle deletion of tracks by the following sequence:

1. Mark tracks to be deleted as invalid,
2. Call [ICinemoPlayer::SignalPlaylistChanged\(\)](#) on all attached players,
3. Handle [GetNextTrack\(\)](#) calls by skipping tracks marked invalid,
4. Delete the tracks.

When implementing [ICinemoTrackProvider](#) methods, the application is entirely responsible for maintaining play order (e.g. sort, shuffle). If the play order of tracks is changed during playback, this is considered a dynamic

change and [ICinemoPlayer::SignalPlaylistChanged\(\)](#) should be called on all attached players. This is because Cinemo players open and queue the next track for gapless playback considerably in advance of the end of the current track (in order to ensure the transition from one track to another is really gapless). When re-ordering tracks, the current *track_id* is expected to remain valid, since *track_id* should be independent of its position or index in the playlist, but next or previous *track_id* is expected to change.

3.23.1 AttachPlayer

Attach a player to the [ICinemoTrackProvider](#) instance.

Syntax

```
CinemoError AttachPlayer(  
    [in]     struct ICinemoPlayer * p  
) ;
```

Parameters

- **p**
The [ICinemoPlayer](#) instance to attach.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is called by [ICinemoPlayer](#) instances when they are created by means of the [CinemoCreatePlayer\(\)](#) function, where the [ICinemoTrackProvider](#) interface is passed as an argument to [CinemoCreatePlayer\(\)](#). There is no defined limit to the number of players which may be attached to an instance of [ICinemoTrackProvider](#). If an application wishes to limit the number of player instances, it may return an error from [AttachPlayer\(\)](#), in which case the [ICinemoPlayer](#) creation will fail and the error code will be returned by the [CinemoCreatePlayer\(\)](#) function.

[ICinemoPlayer](#) instances do not hold a reference count on [ICinemoTrackProvider](#) (they cannot, since [ICinemoTrackProvider](#) does not inherit from [ICinemoUnknown](#) and therefore does not provide reference counting methods). It is the application's responsibility to ensure that [ICinemoTrackProvider](#) instances are not deleted while [ICinemoPlayer](#) instances are still attached.

3.23.2 DetachPlayer

Detach a player from the [ICinemoTrackProvider](#) instance.

Syntax

```
CinemoError DetachPlayer(  
    [in]     struct ICinemoPlayer * p  
) ;
```

Parameters

- **p**
The [ICinemoPlayer](#) instance to detach.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is called by [ICinemoPlayer](#) instances when they are deleted, that is, when [ICinemoUnknown::Release\(\)](#) is called on a player and the reference count becomes zero.

3.23.3 IsValidTrack

Check if a track is valid.

Syntax

```
CinemoError IsValidTrack(  
    [in]      uint64 track_id  
) ;
```

Parameters

- **track_id**

The identifier of the track.

Return value

If the track_id is valid, the method returns *CinemoNoError*. Otherwise, it returns *CinemoFalse*.

Remarks

This method is called by [ICinemoPlayer](#) instances to verify if the currently playing track, or the track queued for gapless playback, is still valid. This may be in response to [ICinemoPlayer::SignalPlaylistChanged\(\)](#).

3.23.4 GetNextTrack

3.23.5 GetPrevTrack

Get the identifier of the next or previous track.

Syntax

```
uint64 GetNextTrack(  
    [in]      uint64 track_id  
) ;  
uint64 GetPrevTrack(  
    [in]      uint64 track_id  
) ;
```

Parameters

- **track_id**

The identifier of the currently playing track.

Return value

A non-zero return value specifies the valid track identifier of the next or previous track. A zero return value specifies that there is no next or previous track.

Remarks

This method may be called frequently by [ICinemoPlayer](#) instances, not only to determine the next or previous track for gapless playback purposes, but also to update the *CINEMO_PUOP_NEXT_TRACK* and *CINEMO_PUOP_PREV_TRACK* mask during playback of the current track. It is the application's responsibility to ensure the function efficiently and quickly returns the required track identifier, taking into account track order and sorting, shuffle, or other parameters unknown to [ICinemoPlayer](#).

3.23.6 GetTrackInfo

Get the URL or additional metadata for a track.

Syntax

```
CinemoError GetTrackInfo(
    [in]     uint64 track_id,
    [out]    struct ICinemoUTF8 ** ppurl,
    [out]    struct ICinemoMetapool ** ppmetapool
);
```

Parameters

- **track_id**

The identifier of the track.

- **ppurl**

To receive the requested [ICinemoUTF8](#) interface.

- **ppmetapool**

To receive the requested [ICinemoMetapool](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Either *ppurl* or *ppmetapool* may be NULL. If *ppurl* is NULL, then the URL is not required and should not be returned. If *ppmetapool* is NULL, then the metapool is not required.

This method is called by [ICinemoPlayer](#) instances prior to playback of a track in order to resolve track identifiers. If a URL is requested, it is the application's responsibility to provide a valid URL. It is not allowed to block for any significant duration in this method, so if the URL is not known at the time the method is called, but can be fetched from e.g. a remote UPnP server, it is allowed to return *CinemoErrorPending* to this method, and subsequently signal [SignalPlaylistChanged\(\)](#) when the URL has been fetched. The [ICinemoPlayer](#) instance will call this method again with the same *track_id*.

This method is also called by [ICinemoPlayer](#) instances in order to obtain additional metadata about a track. Currently only *CINEMO_METANAME_VFS_NAME* and *CINEMO_METANAME_VFS_ICON* are extracted from the returned [ICinemoMetapool](#), and are used to provide additional information to the distributed playback discovery service.

3.23.7 GetRepeat

Get repeat mode.

Syntax

```
CINEMO_PLREPEAT GetRepeat();
```

Return value

Returns the *CINEMO_PLREPEAT* mode of the playlist.

Remarks

See details [ICinemoPlaylist::GetRepeat\(\)](#).

3.23.8 GetOrder

Get play order.

Syntax

```
CINEMO_PLORDER GetOrder();
```

Return value

Returns the *CINEMO_PLORDER* of the playlist.

Remarks

See details [ICinemoPlaylist::GetOrder\(\)](#).

3.23.9 SetRepeatAsync

Set repeat mode of the playlist.

Syntax

```
CinemoError SetRepeatAsync(  
    [in] CINEMO_PLREPEAT repeat  
) ;
```

Parameters

- **repeat**

The playlist repeat mode

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

See details [ICinemoPlaylist::SetRepeat\(\)](#).

3.23.10 SetOrderAsync

Set play order of the playlist.

Syntax

```
CinemoError SetOrderAsync(  
    [in] CINEMO_PLORDER order  
) ;
```

Parameters

- **order**

The play order

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

See details [ICinemoPlaylist::SetOrder\(\)](#).

3.24 ICinemoTrackSelector

The [ICinemoTrackSelector](#) interface provides thread-safe methods to select tracks for playback on an underlying device, for example an Apple or Bluetooth device, without using [ICinemoPlayer](#).

The [ICinemoTrackSelector](#) interface may be obtained by calling [ICinemoUnknown::QueryInterface\(\)](#) on an [ICinemoVFS](#) object which supports it. Support for this interface is indicated by `CINEMO_VFS_FLAG_SELECTABLE` flag returned from [ICinemoVFS::Open\(\)](#), and is currently limited to browse containers on Apple and Bluetooth devices.

Note, it is not required to use this interface when browsing Apple or Bluetooth devices with [ICinemoPlaylist](#). Instead, equivalent functionality is exposed by [ICinemoPlaylist::Select\(\)](#) and [ICinemoPlaylist::SelectTrack\(\)](#) methods.

ICinemoTrackSelector Methods

METHOD	DESCRIPTION
Select	Select for playback on the underlying device
SelectChild	Select a child object for playback on the underlying device

Notes

The [ICinemoTrackSelector](#) interface (and its equivalent [ICinemoPlaylist::Select\(\)](#) and [ICinemoPlaylist::SelectTrack\(\)](#) methods in [ICinemoPlaylist](#)) allow an application to separate the playback and browsing functions on Apple or Bluetooth devices, resulting in the user experience intended by the underlying device protocols.

For such devices, it is recommended to construct a single [ICinemoPlayer](#) instance which only plays the “now playing” track on the device. This player instance may be created, for example, when the device is first inserted, and should persist as long as the device is selected for playback. The player will continuously handle the live audio stream output by the device, and generate metadata updates whenever new metadata is output by the device, for example, when the current playing track changes.

The application may then use, in parallel, [ICinemoPlaylist](#) or [ICinemoVFS](#) methods to browse the device hierarchy and display the resulting tracks in its user interface. If a track, album or playlist is selected for playback, the application may call [ICinemoPlaylist::SelectTrack\(\)](#), which will trigger playback of the selected tracks on the device. There is no need to stop and restart the [ICinemoPlayer](#) instance playing the “now playing” track, which will seamlessly receive the audio stream and metadata updates for the new tracks.

3.24.1 Select

Select the current track or container for playback on the underlying device.

Syntax

```
CinemoError Select(
    [in] const CinemoSelectParams& params
);
```

Parameters

- **params**

`CinemoSelectParams` to specify additional parameters.

Return value

If the method succeeds, it returns `CinemoNoError`. Otherwise, it returns a `CinemoError` code.

Remarks

The behaviour of this method depends on what has been opened by the [ICinemoVFS](#) object implementing this [ICinemoTrackSelector](#) interface. If a single track was opened, then the track will be selected for playback on the underlying device. If, on the other hand, a browse container such as an *album* or *playlist* was opened, then the entire container will be selected for playback on the underlying device.

The `CinemoSelectParams` are defined as follows:

```
typedef struct CinemoSelectParams {
    uint32 flags;
    struct {
        uint32 index;
        uint32 count;
    } range;
} CinemoSelectParams;
```

It is possible to define a range, which is to be selected, when calling this method. If the `CINEMO_SELECTFLAGS_RANGE` bit is set in the `flags` variable, the range will be respected, otherwise it will be ignored.

Depending on the size of the playlist, different scenarios are possible. Here are some examples for a playlist containing five tracks:

- **Index=0 and Count=0**

The entire playlist will be selected.

- **Index=2 and Count=0**

Starting with track 2, all remaining tracks will be selected.

- **Index=2 and Count=2**

Starting with track 2, two tracks will be selected.

- **Index=2 and Count=10**

Starting with track 2, all remaining tracks will be selected (even if count is larger than the playlist count).

- **Index=6 and Count=10**

Cinemo will return *CinemoErrorArguments* because the index is out of bounds.

3.24.2 SelectChild

Select a child object for playback on the underlying device.

Syntax

```
CinemoError SelectChild(
    [in]  uint32 index,
    [in]  const CinemoSelectParams& params
);
```

Parameters

- **index**

Index of the child object to select.

- **params**

`CinemoSelectParams` to specify additional parameters.

Return value

If the method succeeds, it returns `CinemoNoError`. Otherwise, it returns a `CinemoError` code.

Remarks

This method may be called if the [ICinemoVFS](#) implementing this [ICinemoTrackSelector](#) interface has opened a *container* on the device, and it is desired to select one of the container's children for playback.

Note that this method is essentially a shortcut, functionally equivalent to calling [ICinemoVFS::Open\(\)](#) on the child object, followed by [ICinemoTrackSelector::Select\(\)](#). It saves the calling application from having to make an additional [ICinemoVFS::Open\(\)](#) call.

It is possible to define a range, which is to be selected, when calling this method. If the `CINEMO_SELECTFLAGS_RANGE` bit is set in the flags variable, the range will be respected, otherwise it will be ignored.

Depending on the size of the playlist, different scenarios are possible. Here are some examples for the `CinemoSelectParams` for a playlist containing five tracks and a select index of 3:

- **Index=0 and Count=0**

The entire playlist will be selected and playback starts with track 3.

- **Index=2 and Count=0**

Starting with track 2, all remaining tracks will be selected and playback starts with track 3.

- **Index=2 and Count=2**

Tracks 2 and 3 will be selected and playback starts with track 3.

- **Index=4 and Count=2**

Cinemo will return `CinemoErrorArguments` because the select index (3) is not contained in the selection.

- **Index=2 and Count=10**

Starting with track 2, all remaining tracks will be selected (even if count is larger than the playlist count) and playback starts with track 3.

- **Index=6 and Count=10**

Cinemo will return *CinemoErrorArguments* because the index is out of bounds.

3.25 ICinemoPainter

The [ICinemoPainter](#) interface is used for supporting painting in a thread-safe manner by the GUI framework thread. This feature is platform dependent. The [ICinemoPainter](#) interface can only be obtained by calling [ICinemoPlayer::InitPainter\(\)](#) method, so first it is required to obtain an [ICinemoPlayer](#) interface.

In some use cases it's necessary to do the painting from the main thread of the GUI framework like Clutter or Qt. In this case Cinemo will inform the application when painting is needed via the [CinemoUpdateCallback](#). Here the application can inform the GUI framework to invalidate the window and trigger a repaint. From the GUI framework thread the application paint implementation calls [ICinemoPainter::Paint\(\)](#) function using the framework's graphics context.

ICinemoPainter Methods

METHOD	DESCRIPTION
Paint	Platform dependent painting called from GUI framework thread
SetUpdateCallback	Set callback function to receive window content update events

3.25.1 Paint

Platform dependent painting called from GUI framework thread.

Syntax

```
CinemoError Paint(  
    [in]      void * hdc,  
    [in]      const CinemoRect& rc  
) ;
```

Parameters

- **hdc**
Platform dependent parameters for video rendering.
- **rc**
Rectangle needing repaint.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

In case of OpenGL-ES2 platform the **hdc** parameter should point to float[4][4] model view matrix. If **hdc** is NULL then [Paint\(\)](#) will use the GL_VIEWPORT rectangle to construct a model view matrix. **rc** is ignored.

3.25.2 SetUpdateCallback

Set callback function to receive window content update events.

Syntax

```
CinemoError SetUpdateCallback (
    [in]     void * puser,
    [in]     CinemoUpdateCallback pcall
);
```

Parameters

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

The application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The user defined application callback function will be called asynchronously whenever the window needs updating. The application is responsible for informing the GUI framework to trigger a repaint.

To stop receiving update events, a NULL callback function may be specified.

3.26 ICinemoAudioCodec

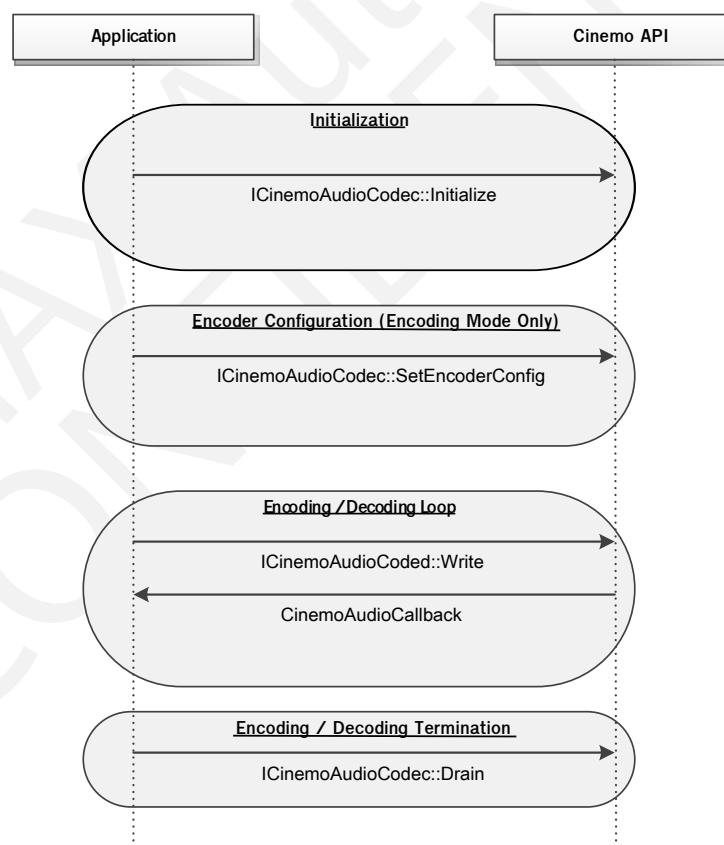
The [ICinemoAudioCodec](#) interface provides direct access to Cinemo audio codecs. This interface is flexible and can support both encoding and decoding operations. Please note that this interface is not thread-safe.

ICinemoAudioCodec Methods

METHOD	DESCRIPTION
Initialize	Initialize codec with media type and callback functions
Write	Write data for encoding or decoding
Drain	Drain data from codec
Flush	Flush and reset codec internal state
SetEncoderConfig	Set encoder configuration
GetEncoderInfo	Get encoder information

Notes

The call sequence required for MP3 or AAC encoding is shown here:



3.26.1 Initialize

Initialize codec with media type and callback functions.

Syntax

```
CinemoError Initialize(  
    [in] void * puser,  
    [in] CinemoAudioCallback pcall,  
    [in] const CinemoMediaType& input,  
    [in] const CinemoMediaType& output  
) ;
```

Parameters

- **puser**
User-specified parameter.
- **pcall**
Pointer to the callback function.
- **input**
Source media type.
- **output**
Destination media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This call initializes the decoder or encoder. When decoding audio content the output media type can be left empty to receive the original decoder content or it can specify a PCM format to be used as output. Cinemo Media Engine is using a function callback for sending back the encoded or decoded data. In case of AAC encoding, data will be actual encoded audio frames, while in the MP3 decoding case, data will be in the PCM format.

3.26.2 Write

Write data for encoding or decoding.

Syntax

```
CinemoError Write(  
    [in] const void * psrc,  
    [in] uint32 nbytes,  
    [in] pts70mhz npts,  
    [in] uint32 nptsvalid  
) ;
```

Parameters

- **psrc**
Source buffer.
- **nbytes**
Size of the source buffer (in bytes).
- **npts**
Timestamp (optional).
- **nptsvalid**
Indicates if timestamp is valid.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This call passes an input audio packet to the codec (MP3 decoder or AAC encoder). The output callback will be called when the decoding or encoding of one unit is ready. Timestamp information is optional.

3.26.3 Drain

Drain data from codec.

Syntax

```
CinemoError Drain();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This call drains the remaining data held in internal buffers. This function should be called at the end of the audio encoding/decoding session.

3.26.4 Flush

Flush and reset codec internal state.

Syntax

```
CinemoError Flush();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This call flushes the internal codec state.

3.26.5 SetEncoderConfig

Set encoder configuration.

Syntax

```
CinemoError SetEncoderConfig(  
    [in]  const CinemoEncoderConfig& config  
) ;
```

Parameters

- **config**

Encoder configuration parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method changes the configuration of the encoder. The encoder configuration parameters are defined by the following structure:

```
typedef struct {  
    uint32 bitrate;           /* Encoder bit rate */  
    bool is_cbr;             /* True if constant bitrate encoding */  
} CinemoEncoderConfig;
```

Currently only setting the encoder bitrate and whether to use CBR are supported. This structure may be extended in future, so ensure to zero all parameters before passing to [SetEncoderConfig\(\)](#).

3.26.6 GetEncoderInfo

Get encoder information.

Syntax

```
CinemoError GetEncoderInfo(  
    [out] CinemoEncoderInfo& info  
) ;
```

Parameters

- **attrib**

Return encoder information.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, the encoder information structure will be initialized with the current encoder information. The *CinemoEncoderInfo* structure is defined here:

```
typedef struct {  
    uint64 sample_count;      /* Encoder number of samples */  
    uint32 delay;            /* Encoder delay (in samples) */  
    uint32 frame_duration;   /* Encoder frame duration (in samples) */  
} CinemoEncoderInfo;
```

3.27 ICinemoVFS

The [ICinemoVFS](#) interface provides low-level metadata extraction functions. It may be used by applications which require metadata independently of any Cinemo playback or playlist functionality. [ICinemoVFS](#) is capable of extracting metadata from folders, files, and remote servers, including UPnP content directory services and discovery. It is used extensively by Cinemo test applications to browse local and remote file systems, and to generate video thumbnails along with other kinds of metadata. This interface is thread-safe when considering the remarks in the [Notes](#) section.

ICinemoVFS Methods

METHOD	DESCRIPTION
Open	Open a URL and obtain metadata
OpenSource	Open a source object and obtain metadata
Watch	Watch the currently open URL for metadata changes
GetAudioStream	Get audio stream information
GetVideoStream	Get video stream information
GetSubpictureStream	Get subpicture stream information
GetMediaInfo	Get media information
GetTitleInfo	Get title information
GetChapterInfo	Get chapter information
Cancel	Cancel any Open or Watch calls
Enable	Enable any Open or Watch calls after a Cancel
SetThumbFormat	Set parameters for thumbnail generation
Close	Close the current URL

Notes

All metadata extraction methods are *synchronous* and *blocking*: the calling application is responsible for creating threads or other synchronization objects to avoid, for example, hanging up the GUI while calling these methods. Depending on the object being queried, metadata extraction can take **significant time**: for example, CD_TEXT extraction from a CD may require spinning up the disc or waiting on ATAPI timeouts for scratched discs, and other kinds of objects may require waiting on HTTP requests to remote servers.

Calls to the methods [Open\(\)](#), [OpenSource\(\)](#), [Watch\(\)](#) and [Close\(\)](#) shall be serialized on the caller side.

3.27.1 Open

Open a URL and obtain metadata.

Syntax

```
CinemoError Open(
    [in] const char * szurl,
    [in] uint32 flags,
    [out] CinemoVFSAttributes& attrs,
    [out] struct ICinemoMetapool ** pp
);
```

Parameters

- **szurl**

UTF-8 zero terminated URL.

- **flags**

Bitmask of *CINEMO_VFS_OPEN* flags to control metadata extraction. For more details please refer to the [Flags](#) section.

- **attrs**

Return attributes of the opened VFS.

- **pp**

The address of a pointer to receive the requested [ICinemoMetapool](#).

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

On successful return, the *CinemoVFSAttributes* will be filled with the attributes of the URL which was opened. The VFS attributes are defined by the following structure:

```
typedef struct {
    uint32 type;           /* bitmask of CINEMO_VFS_TYPE */
} CinemoVFSAttributes;
```

On successful return, the reference count of the returned [ICinemoMetapool](#) interface will be automatically incremented. Its [ICinemoUnknown::Release\(\)](#) method should be called when it is no longer required. The [ICinemoMetapool](#) will contain all known metadata about the opened URL.

Flags

The following flags can be used for creating a bitmask which can be passed as the *flags* parameter.

- **CINEMO_VFS_OPEN_FOLDER**

The supplied URL points to a folder.

• CINEMO_VFS_OPEN_THUMBNAIL_WHEN_NO_COVERART

If no coverart is provided for a media item, Cinemo will try to generate a thumbnail by extracting a video frame from the media item.

• CINEMO_VFS_OPEN_THUMBNAIL_FROM_COVERART

If coverart is provided for a media item, Cinemo will scale it to generate a thumbnail.

• CINEMO_VFS_OPEN_THUMBNAIL_ALWAYS

Cinemo will generate a thumbnail preferably based on the coverart, alternatively by extracting a video frame.

• CINEMO_VFS_OPEN_THUMBNAIL_ALWAYS_IGNORE_COVERART

Cinemo will always try to generate the thumbnail by extracting a video frame from the media item, regardless of coverart being available.

• CINEMO_VFS_OPEN_DEVICE

If this ICinemoVFS instance is being closed, also release device related resources such as exclusive USB access.

• CINEMO_VFS_OPEN_METADATA_ONLY

Optimization hint for only metadata extraction in case of a media item. As example, sample tables of MP4 and AVI are not read. This might improve speed and save memory. Please note, that thumbnail generation might not work with this flag enabled.

3.27.2 OpenSource

Open a source object and obtain metadata.

Syntax

```
CinemoError Open(
    [in] struct ICinemoUnknown * psource,
    [in] uint32 flags,
    [out] CinemoVFSAttributes& attrs,
    [out] struct ICinemoMetapool ** pp
);
```

Parameters

- **psource**

Pointer to a user-provided source object.

- **flags**

Bitmask of *CINEMO_VFS_OPEN* flags to control metadata extraction. For more details please refer to the [Flags](#) section.

- **attrs**

Return attributes of the opened VFS.

- **pp**

The address of a pointer to receive the requested [ICinemoMetapool](#).

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

On successful return, the *CinemoVFSAttributes* will be filled with the attributes of the source object which was opened. The VFS attributes are defined by the following structure:

```
typedef struct {
    uint32 type;           /* bitmask of CINEMO_VFS_TYPE */
} CinemoVFSAttributes;
```

On successful return, the reference count of the returned [ICinemoMetapool](#) interface will be automatically incremented. Its [ICinemoUnknown::Release\(\)](#) method should be called when it is no longer required. The [ICinemoMetapool](#) will contain all known metadata about the opened source object.

3.27.3 Watch

Watch the currently open URL for metadata changes.

Syntax

```
CinemoError Watch(  
    [out] struct ICinemoMetapool ** pp  
    [in] uint32 timeout_ms  
) ;
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoMetapool](#).

- **timeout_ms**

Timeout in milliseconds, or 0 for infinite timeout (use [Cancel\(\)](#) method to unblock).

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is used to watch for metadata changes. It should only be called if [Open\(\)](#) returned success and the returned VFS type indicated *CINEMO_VFS_TYPE_WATCHABLE*. Metadata changes which occur after the [Open\(\)](#) call while not yet watching the VFS will be returned with the following [Watch\(\)](#) call.

On successful return, the returned [ICinemoMetapool](#) interface will contain all known metadata about the URL (not just the changes). The reference count of the returned [ICinemoMetapool](#) interface will be automatically incremented. Its [ICinemoUnknown::Release\(\)](#) method should be called when it is no longer required. The [ICinemoMetapool](#) will contain all known metadata about the opened URL.

Using non-zero time-out is inferior to [Cancel\(\)](#) unblocking. *CinemoErrorTimeout* is returned when time-out happened because of *timeout_ms* parameter. Device or network connection related *CinemoErrorTimeout* is converted to *CinemoErrorFailed* if *timeout_ms* parameter is non-zero.

3.27.4 GetAudioStream

Get audio stream information of the current title.

Syntax

```
CinemoError GetAudioStream(  
    [in]  uint32 audio_id,  
    [out] CinemoMediaType &media  
) ;
```

Parameters

- **audio_id**
Audio stream identifier.
- **media**
Returned audio stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method can only be called after a VFS has been opened. If the VFS is playable, indicated by the *CINEMO_VFS_FLAG_PLAYABLE* flag, the audio stream information of the requested audio stream is available in the media structure.

The returned media type will only be valid for “files” or single-title media. For media containing several titles, the behaviour is undefined and depends on the navigator. Instead the [ICinemoPlayer::GetAudio\(\)](#) function should be used.

3.27.5 GetVideoStream

Get video stream information of the current title.

Syntax

```
CinemoError GetVideoStream(  
    [in]  uint32 video_id,  
    [out] CinemoMediaType &media  
) ;
```

Parameters

- **video_id**

Video stream identifier.

- **media**

Returned video stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method can only be called after a VFS has been opened. If the VFS is playable, indicated by the *CINEMO_VFS_FLAG_PLAYABLE* flag, the video stream information of the requested video stream is available in the media structure.

The returned media type will only be valid for “files” or single-title media. For media containing several titles, the behaviour is undefined and depends on the navigator. Instead the [ICinemoPlayer::GetAngle\(\)](#) function should be used.

3.27.6 GetSubpictureStream

Get subpicture stream information of the current title.

Syntax

```
CinemoError GetSubpictureStream(  
    [in]  uint32 subpicture_id,  
    [out] CinemoMediaType &media  
) ;
```

Parameters

- **subpicture_id**

Subpicture stream identifier.

- **media**

Returned subpicture stream media type.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method can only be called after a VFS has been opened. If the VFS is playable, indicated by the *CINEMO_VFS_FLAG_PLAYABLE* flag, the subpicture stream information of the requested subpicture stream is available in the media structure.

The returned media type will only be valid for “files” or single-title media. For media containing several titles, the behaviour is undefined and depends on the navigator. Instead the [ICinemoPlayer::GetSubpicture\(\)](#) function should be used.

3.27.7 GetMediaInfo

Get media information about the current medium.

Syntax

```
CinemoError GetMediaInfo(  
    [out] CinemoMediaInfo &mediainfo  
);
```

Parameters

- **mediainfo**

Returned media information.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method can only be called after a VFS has been opened. If the VFS is playable, indicated by the *CINEMO_VFS_FLAG_PLAYABLE* flag, the returned media information will contain the title number and encryption flag for the medium.

3.27.8 GetTitleInfo

Get information about a specific title.

Syntax

```
CinemoError GetTitleInfo(  
    [in]  uint32 title_id,  
    [out] CinemoTitle &title  
) ;
```

Parameters

- **title_id**
Title identifier.
- **title**
Returned title information.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method can only be called after a VFS has been opened. If the VFS is playable, indicated by the *CINEMO_VFS_FLAG_PLAYABLE* flag, the title information of the requested title_id will be available in the title structure.

3.27.9 GetChapterInfo

Get information about a specific chapter.

Syntax

```
CinemoError GetChapterInfo(  
    [in]  uint32 title_id,  
    [in]  uint32 chapter_id,  
    [out] CinemoChapter &chapter  
) ;
```

Parameters

- **title_id**

Title identifier.

- **chapter_id**

Chapter identifier.

- **chapter**

Returned chapter information.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method can only be called after a VFS has been opened. If the VFS is playable, indicated by the *CINEMO_VFS_FLAG_PLAYABLE* flag, the chapter information of the requested *chapter_id* and *title_id* will be available in the chapter structure.

3.27.10 Cancel

3.27.11 Enable

Cancel or enable all [Open\(\)](#) and [Watch\(\)](#) methods.

Syntax

```
CinemoError Cancel();  
CinemoError Enable();
```

Return value

If the method succeeds, it returns *CinemoNoError*. The method always succeeds.

Remarks

This method may be called to asynchronously cancel any thread blocked on [Open\(\)](#) or [Watch\(\)](#) calls. In this case, [Open\(\)](#) or [Watch\(\)](#) will return *CinemoErrorCancel*.

Note the cancel operation is sticky – after calling [Cancel\(\)](#), the [ICinemoVFS](#) object will remain in a cancelled state until [Enable\(\)](#) is called. This is to avoid potential race conditions. The correct sequence of calls to unblock a thread blocked in [ICinemoVFS](#) is as follows:

1. Call [Cancel\(\)](#)
2. Wait for your thread which calls [Open\(\)](#) or [Watch\(\)](#) to exit
3. Call [Enable\(\)](#)

3.27.12 SetThumbFormat

Set parameters for thumbnail generation.

Syntax

```
CinemoError SetThumbnailFormat(
    [in] const CinemoThumbFormat& thumb
);
```

Parameters

- **thumb**

Specify maximum width, height and encoded image format.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The method may be used to specify the parameters for thumbnail generation. Thumbnails will only be generated if the [Open\(\)](#) method is called with one of the following flags:

- *CINEMO_VFS_OPEN_THUMBNAIL_WHEN_NO_COVERART*
- *CINEMO_VFS_OPEN_THUMBNAIL_ALWAYS*

Thumbnail parameters are specified by the *CinemoThumbFormat* structure:

```
typedef struct {
    uint32 cx;
    uint32 cy;
    CINEMO_MEDIASUBTYPE subtype;
} CinemoThumbFormat;
```

The generated thumbnail will be scaled and encoded to the given subtype image format, such that its aspect ratio is preserved, and its width or height does not exceed the specified parameters. If one or the other of cx, cy are set to zero, then no restriction is implied. The following image formats are currently supported:

- *CINEMO_MEDIASUBTYPE_JPEG*
- *CINEMO_MEDIASUBTYPE_PNG*
- *CINEMO_MEDIASUBTYPE_BMP*

The method will return with *CinemoErrorArguments* if any other subtype is specified.

3.27.13 Close

Close the current URL.

Syntax

```
CinemoError Close();
```

Return value

The method returns *CinemoNoError*.

Remarks

This method frees all resources but does not release the interface.

3.28 ICinemoMM

The [ICinemoMM](#) interface provides access to Cinemo Media Management server. It can be created using the `CinemoCreateMM()` function defined in `cinemo_mm.h`. Please note that this interface is not thread-safe.

This interface may be used either to create a local MM server, or to connect to and remotely administer an existing MM server on the network. Once connected, the behaviour of the interface is identical whether or not the server is local or remote.

The interface may be used to create or delete volumes, mount or dismount volumes, and to edit content within volumes. It may be used for *journaling* purposes to track changes to volumes and their contents, for example if the volume database needs to be replicated elsewhere. It may be used for or *metadata normalization* purposes, for example if an application wishes to supply alternative metadata for indexed content, e.g. as obtained from a third-party SDK.

ICinemoMM Methods

METHOD	DESCRIPTION
CreateServer	Create an MM server
ConnectToServer	Connect to an existing MM server, either local or remote
GetServerURL	Returns the Cinemo MM server URL.
Cancel	Cancel all operations
Enable	Enable all operations after a call to Cancel
CreateVirtualFileServer	Create a file server to serve local content on demand
AddVolumeGroup	Add a volume group
AddIndexedVolume	Add an indexed volume
AddVirtualVolume	Add a virtual volume
AddVirtualFile	Add a virtual file
AddVirtualFolder	Add a virtual folder
MountVolume	Mount a volume
DismountVolume	Dismount a volume
ReindexVolume	Re-index a volume
MountVolumes	Mount all volumes matching the specified attributes
DismountVolumes	Dismount all volumes matching the specified attributes
ReindexVolumes	Re-index all volumes matching the specified attributes
EditNode	Edit a file, folder, volume or volume group
RemoveNode	Remove a file, folder, volume or volume group
PauseVolumeIndexer	Pause or resume indexing of a volume or volume group
BrowseMetadata	Read node metadata
BrowseDirectChildren	Read node child metadata
Search	Returns the result of a search

METHOD	DESCRIPTION
<u>SetOption</u>	Set configuration option
<u>GetOption</u>	Get configuration option
<u>GetFieldMap</u>	Get Metaname field mapping table
<u>GetHierarchyNodes</u>	Get hierarchy containers for node
<u>GetParentNodes</u>	Get parent containers for node
<u>StartVolumeEvents</u>	Start receiving volume events
<u>StopVolumeEvents</u>	Stop receiving volume events
<u>StartNodeEvents</u>	Start receiving node events
<u>StopNodeEvents</u>	Stop receiving node events
<u>StartIndexingEvents</u>	Start receiving indexing events
<u>StopIndexingEvents</u>	Stop receiving indexing events
<u>RegisterQueryHandler</u>	Register a query expression handler
<u>UnregisterQueryHandler</u>	Unregister a query handler
<u>RegisterMetadataHandler</u>	Register a metadata expression handler
<u>UnregisterMetadataHandler</u>	Unregister a metadata request handler
<u>SetCachedRevision</u>	Store journaling revision
<u>GetCachedRevision</u>	Retrieve journaling revision

3.28.1 CreateServer

Create an MM server.

Syntax

```
CinemoError CreateServer(  
    [in] const char * szdbfolder  
) ;
```

Parameters

- **szdbfolder**

Specify the directory where MM database files are stored.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

On successful return, a new instance of Cinemo Media Management server will be instantiated and available on the network. It can be accessed either locally or remotely by other Cinemo API's. For example, it is discoverable as a UPnP media server by [ICinemoVFS](#), and it can be searched or browsed by [ICinemoVFS](#) or [ICinemoPlaylist](#) API's. The server will remain active until the [ICinemoMM](#) interface is released, after which all its resources will be freed.

On successful return from [CreateServer\(\)](#), it is possible to call all other methods on this interface. It is not necessary to call [ConnectToServer\(\)](#).

3.28.2 ConnectToServer

Connect to an existing MM server, either local or remote.

Syntax

```
CinemoError ConnectToServer(  
    [in] const char * szurl  
) ;
```

Parameters

- **szurl**

The URL of the MM server.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The URL of the MM server, if not explicitly known, can be obtained by calling [ICinemoVFS::Open\(\)](#) in UPnP discovery mode (see section 0). It is returned as the VFS_PATH attribute of discovered media servers. The following is an example URL for a Cinemo Media Management server running locally on port number 49158:

```
upnp://127.0.0.1:49158/0/MediaServer/DeviceDesc.xml
```

After calling [ConnectToServer\(\)](#), it is possible to call all other methods on this interface.

3.28.3 GetServerURL

Returns the Cinemo MM server URL.

Syntax

```
CinemoError GetServerURL(  
    [out] struct ICinemoUTF8 ** pp  
) ;
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The returned URL will either be the URL passed to [ConnectToServer\(\)](#), or the localhost address and port number of the server created by [CreateServer\(\)](#).

3.28.4 Cancel

3.28.5 Enable

Cancel or enable [ICinemoMM](#) methods.

Syntax

```
CinemoError Cancel();  
CinemoError Enable();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

[Cancel\(\)](#) will unblock any call to all [ICinemoMM](#) methods (with the exception of [ICinemoMM::CreateServer\(\)](#), which is a non-blocking method). Like all other Cinemo API's, the [Cancel\(\)](#) operation is sticky – after calling [Cancel\(\)](#), the [ICinemoMM](#) object will remain in a cancelled state until [Enable\(\)](#) is called. This is to avoid potential race conditions.

If a method is unblocked by [Cancel\(\)](#), it will return *CinemoErrorCancel* error code.

Since all [ICinemoMM](#) methods involve connecting by HTTP or otherwise to a local or remote Cinemo Media Management server, depending on network conditions, they may potentially block for a long time, or infinitely. Correct use of [ICinemoMM](#) methods requires the calling application to maintain threads and implement synchronization strategies to avoid hang ups.

3.28.6 CreateVirtualFileServer

Create a file server to serve local content on demand.

Syntax

```
CinemoError CreateVirtualFileServer(
    [in] void * puser,
    [in] CinemoMMFileServerCallback pcall,
    [in] const char * szurl
);
```

Parameters

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

- **szurl**

IP address and port for binding the file server.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

[CreateVirtualFileServer\(\)](#) will instantiate a local file server (currently implemented as an HTTP server) for serving virtual files on demand. Virtual files are uniquely identified by an application defined key passed to [AddVirtualFile\(\)](#). The file server will be created on the specified port, or a random port if the specified port is zero. Note – do *not* use a random port for a server which serves files to *persistent* virtual volumes, otherwise it will not be possible to serve the file after a restart. For persistent virtual volumes, [CreateVirtualFileServer\(\)](#) should be called with the same port number every time the system is restarted.

An application defined callback function will be called whenever the virtual file is requested by the server. The callback function is of the following form:

```
CinemoError CinemoMMFileServerCallback(
    [in] void * puser,                                /* user data */
    [in] char * szkey,                               /* key specified in AddVirtualFile() */
    [out] struct ICinemoUTF8 ** ppmimetype,        /* return mimetype */
    [out] struct ICinemoFile ** ppfile);            /* return file interface */
```

- **puser**

The user defined parameter passed to [CreateVirtualFileServer\(\)](#).

- **szkey**

The key value passed to [AddVirtualFile\(\)](#).

- **ppmimetype**

Return the CONTENT-TYPE of the file.

- **ppfile**

Return an [ICinemoFile](#) interface for reading the file.

The application is responsible for returning an [ICinemoFile](#) interface which can be used to read the file identified by the given key. An example implementation can be found in the Cinemo SDK SampleMM.cpp source file.

The szurl parameter accepts values with the syntax

```
tcp://ip:port
```

Special values can be used to identify all IP addresses or ports as well as differentiate between IPV4 and IPV6 mode. See the documentation of *CINEMO_OPTION_MM_SERVER_URL* for examples for these special values. Care has to be taken that the IP address and port are accessible by the Media Management server as well as by all Media Management clients.

3.28.7 AddVolumeGroup

Add a volume group.

Syntax

```
CinemoError AddVolumeGroup(
    [in] uint64 npid,                                /* parent id */
    [in] const char * szhierarchy,                  /* hierarchy identifier */
    [in] const char * szuuid,                        /* unique id */
    [in] const char * szname,                        /* name */
    [out] CinemoMMNodeAttributes& attrs           /* return attributes */
);
```

Parameters

- **npid**

The parent node id.

- **szhierarchy**

The MM container hierarchy to create in this volume group.

- **szuuid**

The unique ID of the volume group.

- **szname**

The name of the volume group.

- **attrs**

Return the volume group attributes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[AddVolumeGroup\(\)](#) will either create a new volume group, or return the attributes of an existing volume group which matches the specified parameters. It is not possible to create more than one volume group on the same parent with the same parameters. However, volume groups created as children of different parent nodes may share identical parameters.

The *npid* argument specifies the ID of an existing volume group node. Volume groups can only be created as children of other volume groups. Volume groups may be statically defined in the Cinemo MM configuration file. The Cinemo MM root node is a volume group.

The *szname* parameter must not be NULL. All volume groups have must have a name.

The *szuuid* parameter is optional. If specified, the unique ID will be assigned to the volume group, and any attempt to create another volume group with the same parent and unique ID parameters will instead return the existing volume group. If set to NULL, the *szname* parameter will be used as the unique ID.

The *szhierarchy* parameter is optional. If specified, it must be the identifier of a set of hierarchy rules defined in a *cinemo_mm_hierarchy* section of the configuration XML.

3.28.8 AddIndexedVolume

Add an indexed volume.

Syntax

```
CinemoError AddIndexedVolume(  
    [in]  uint64 npid,                      /* parent id */  
    [in]  const char * szmountpath,          /* mountpath */  
    [in]  const char * szuuid,                /* UUID */  
    [in]  const char * szname,                /* name */  
    [in]  const char * sztype,                /* type */  
    [out] CinemoMMNodeAttributes& attrs     /* return attributes */  
) ;
```

Parameters

- **npid**

The parent node id.

- **szmountpath**

The pathname of a directory or mass storage device to index.

- **szuuid**

The unique ID of the volume.

- **szname**

The name of the volume.

- **sztype**

The user-defined type of the volume.

- **attrs**

Return the volume attributes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[AddIndexedVolume\(\)](#) will add a new indexed volume to the MM server. After calling this method, the indexed volume will be visible in the UPnP hierarchy, but not yet mounted. The volume will not be indexed until it is mounted with [MountVolume\(\)](#).

When the volume is mounted, Cinemo will automatically index the contents of the specified *szmountpath*, recursively adding all folders and files to the persistent database. Multiple indexing passes will be made to extract filenames, metadata, and thumbnails. The specified mount path must be local to the MM server, e.g.

the name of a directory containing media content, or the path to a USB or other removable storage device already mounted by the operating system.

The *npid* argument specifies the ID of an existing volume group node. Indexed volumes can only be created as children of volume groups. Volume groups may be statically defined in the MM configuration file, or may be dynamically created by means of [AddVolumeGroup\(\)](#). The location of an indexed volume in the volume group hierarchy is persistent, i.e. stored in the volume database for each volume, and volume groups containing indexed volumes will be automatically re-created if Cinemo MM is restarted.

The *szzuid* argument is optional. If NULL, then depending on MM configuration options, Cinemo will attempt to automatically determine the UUID from the supplied mountpath. Cinemo will not allow creation of more than one volume with the same UUID. If another volume with the same UUID already exists, then [AddIndexedVolume\(\)](#) will do nothing, and will return success with the attributes of the existing volume. This is to avoid re-indexing volumes which have already been indexed, even if *szmountpath* has changed.

The *szname* argument is optional. If specified, the volume will appear with this name in the UPnP hierarchy; otherwise, the name will fall back to the volume UUID, if specified, or the mount path.

The *sztype* argument is optional. If specified, the volume will be assigned the user-defined type, which will be returned as the metadata item *CINEMO_METANAME_VFS_UPNP_VOLUME_TYPE* in search or browse results.

3.28.9 AddVirtualVolume

Add a virtual volume.

Syntax

```
CinemoError AddVirtualVolume(
    [in] uint64 npid,                                /* parent id */
    [in] const char * szpersistent,                  /* volume is persistent or not */
    [in] const char * szuuid,                        /* UUID */
    [in] const char * szname,                        /* name */
    [in] const char * sztype,                        /* type */
    [out] CinemoMMNodeAttributes& attrs           /* return attributes */
);

```

Parameters

- **npid**

The parent node id.

- **szpersistent**

Specify whether the volume should be persistent.

- **szuuid**

The unique ID of the volume.

- **szname**

The name of the volume.

- **sztype**

The user-defined type of the volume.

- **attrs**

Return the volume attributes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[AddVirtualVolume\(\)](#) method will add a persistent or non-persistent virtual volume to the MM server. After calling this method, the virtual volume will be visible in the UPnP hierarchy, but not yet mounted. The volume will not be writable until it is mounted with [MountVolume\(\)](#). After the volume has been mounted, an application may populate the volume by calling [AddVirtualFile\(\)](#) or [AddVirtualFolder\(\)](#) methods, specifying the returned volume ID.

The *npid* argument must specify the identifier of an existing volume group node. Virtual volumes can only be created as children of volume groups. Volume groups can be statically defined in the MM configuration file, or may be dynamically created by means of [AddVolumeGroup\(\)](#). The location of a persistent virtual volume in the volume group hierarchy is also persistent, i.e. stored in the volume database, and volume groups containing persistent virtual volumes will be automatically re-created if Cinemo MM is restarted.

The *szpersistent* argument indicates whether the virtual volume should be persistent or not. Currently this argument is interpreted as a simple *boolean*, where NULL means non-persistent, and any non-NUL value means persistent. This may be changed in future Cinemo versions to allow specifying additional attributes of persistent virtual volumes. It is recommended to pass the string value "true" for persistent virtual volumes.

The *szzuid* argument is optional. Cinemo will not allow creation of more than one volume with the same UUID. If another volume with the same UUID already exists, then [AddVirtualVolume\(\)](#) will do nothing, and will return success with the attributes of the existing volume.

The *szname* argument is optional. If specified, the volume will appear with this name in the UPnP hierarchy; otherwise, the name will fall back to the volume UUID.

The *sztype* argument is optional. If not NULL, the volume will be assigned the user-defined type, which will be returned as the metadata item *CINEMO_METANAME_VFS_UPNP_VOLUME_TYPE* in search or browse results. The volume type is persistent.

3.28.10 AddVirtualFile

Add a virtual file to a virtual volume.

Syntax

```
CinemoError AddVirtualFile(
    [in] uint64 pid,                      /* parent id */
    [in] const char * szkey,                /* key for CinemoMMFileServerCallback function */
    [in] struct ICinemoMetapool * ppool,   /* metadata */
    [out] CinemoMMNodeAttributes& attrs   /* return attributes */
);
```

Parameters

- **pid**

Parent node identifier.

- **szkey**

Optional application-defined key to uniquely identify the file.

- **ppool**

Metadata describing the virtual file.

- **attrs**

Return the file attributes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[AddVirtualFile\(\)](#) will add a new virtual file to a virtual volume. The virtual volume must already have been mounted by a call to [MountVolume\(\)](#) or [MountVolumes\(\)](#). The parent node identifier must be that of a virtual volume or folder created by a previous call to [AddVirtualVolume\(\)](#) or [AddVirtualFolder\(\)](#).

If *szkey* is not NULL, it must uniquely identify the virtual file. The key will be passed to the application-defined *CinemoMMFileServerCallback* function specified in [CreateVirtualFileServer\(\)](#). This callback function is responsible for constructing an [ICinemoFile](#) interface which can be used to read the file identified by the given key. An exception to this rule is if *szkey* is prefixed with the "file://" protocol, in which case it is not required to serve the file by means of [CreateVirtualFileServer\(\)](#). In this case, Cinemo MM will interpret the key as the *absolute path* of a file accessible to and readable by the Cinemo MM server, and will serve the file itself.

3.28.11 AddVirtualFolder

Add a virtual folder to a virtual volume.

Syntax

```
CinemoError AddVirtualFolder(  
    [in]  uint64 pid,                      /* parent id */  
    [in]  const char * szname,              /* folder name */  
    [out] CinemoMMNodeAttributes& attrs   /* return attributes */  
) ;
```

Parameters

- **pid**

Parent node identifier.

- **szname**

Name of the folder.

- **attrs**

Return the virtual folder attributes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[AddVirtualFolder\(\)](#) will add a new virtual folder to a virtual volume. The virtual volume must already have been mounted by [MountVolume\(\)](#) or [MountVolumes\(\)](#). The parent node identifier must be that of a virtual volume or folder created by a previous call to [AddVirtualVolume\(\)](#) or [AddVirtualFolder\(\)](#).

3.28.12 MountVolume

Mount a volume.

Syntax

```
CinemoError MountVolume(  
    [in]  uint64 id  
);  
/* volume id */
```

Parameters

- **id**

The identifier of the volume.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[MountVolume\(\)](#) will mount an existing indexed or virtual volume. If the volume is already mounted, then no action will be taken. If the volume is an indexed volume, then the indexing process will start, and all media content in the volume will be immediately available and reflected in browse and search results. If the volume is a virtual volume, then all its children will become visible and reflected in browse and search results.

3.28.13 DismountVolume

Dismount a volume.

Syntax

```
CinemoError DismountVolume(  
    [in]  uint64 id  
);  
/* volume id */
```

Parameters

- **id**

The identifier of the volume.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[DismountVolume\(\)](#) method will dismount a previously mounted volume. If the volume is already dismounted, no action will be taken. Otherwise, all children of the volume will be removed from the MM hierarchy and from search and browse results. All resources allocated by the volume will be freed – for example, RAM, open file handles, etc. The persistent state of the volume will remain unchanged.

3.28.14 ReindexVolume

Re-index a volume.

Syntax

```
CinemoError ReindexVolume(  
    [in]  uint64 id  
);  
/* volume id */
```

Parameters

- **id**

The identifier of the volume.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[ReindexVolume\(\)](#) method will re-index a mounted volume which supports re-indexing. The indexing process is restarted, and changes in the media content of the volume will be reflected in browse and search results. For volumes which do not support re-indexing, no action will be taken.

3.28.15 MountVolumes

Mount all volumes matching the specified attributes.

Syntax

```
CinemoError MountVolumes(  
    [in] const char * szmountpath,           /* by mount path */  
    [in] const char * szuuid,                /* by uuid */  
    [in] const char * sztype                /* by type */  
) ;
```

Parameters

- **szmountpath**

The mount path of the volume.

- **szuuid**

The UUID of the volume.

- **sztype**

The type of the volume.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[MountVolumes\(\)](#) will mount all existing indexed or virtual volumes which match the given attributes. If the specified volumes are already mounted, then no action will be taken. Any attribute specified as NULL will not be matched, therefore, [MountVolumes\(NULL,NULL,NULL\)](#) will mount all existing volumes.

3.28.16 DismountVolumes

Dismount all volumes matching the specified attributes.

Syntax

```
CinemoError DismountVolumes(  
    [in] const char * szmountpath,           /* by mount path */  
    [in] const char * szuuid,                /* by uuid */  
    [in] const char * sztype                /* by type */  
) ;
```

Parameters

- **szmountpath**

The mount path of the volume.

- **szuuid**

The UUID of the volume.

- **sztype**

The type of the volume.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[DismountVolumes\(\)](#) will dismount all existing indexed or virtual volumes which match the given attributes. If the specified volumes are already dismounted, then no action will be taken. Any attribute specified as NULL will not be matched, therefore, [DismountVolumes\(NULL,NULL,NULL\)](#) will dismount all existing volumes.

3.28.17 ReindexVolumes

Re-index all volumes matching the specified attributes.

Syntax

```
CinemoError ReindexVolumes(  
    [in] const char * szmountpath,           /* by mount path */  
    [in] const char * szuuid,                /* by uuid */  
    [in] const char * sztype                /* by type */  
) ;
```

Parameters

- **szmountpath**

The mount path of the volume.

- **szuuid**

The UUID of the volume.

- **sztype**

The type of the volume.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[ReindexVolumes\(\)](#) will re-index all mounted volumes which support re-indexing and match the given attributes. The indexing process is restarted, and changes in the media content of the volume will be reflected in browse and search results. For volumes which do not support re-indexing, no action will be taken. Any attribute specified as NULL will not be matched, therefore, [ReindexVolumes\(NULL,NULL,NULL\)](#) will re-index all existing volumes.

3.28.18 EditNode

Edit the metadata of a file, folder, volume or volume group.

Syntax

```
CinemoError EditNode(  
    [in]  uint64 id,                      /* node id */  
    [in]  struct ICinemoMetapool * ppool   /* node metadata */  
) ;
```

Parameters

- **id**

The node identifier.

- **ppool**

The metadata to change.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[EditNode\(\)](#) will modify the metadata fields of files, folders, volumes or volume groups in the MM node hierarchy. This method may be used, for example, to perform metadata normalization. All changes will be immediately visible to connected MM clients by means of dynamic updates.

3.28.19 RemoveNode

Remove a file, folder, volume or volume group.

Syntax

```
CinemoError RemoveNode (
    [in]  uint64 id
) ;                                     /* node id */
```

Parameters

- **id**

The node identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[RemoveNode\(\)](#) will remove the node from the MM node hierarchy. All persistent database files and resources associated with the node will be deleted.

3.28.20 PauseVolumeIndexer

Pause or resume indexing of a volume or volume group.

Syntax

```
CinemoError PauseVolumeIndexer(  
    [in]  uint64 id,                      /* node id */  
    [in]  uint32 pause                    /* pause state */  
) ;
```

Parameters

- **id**
The volume or volume group identifier.
- **pause**
The indexing pause state.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

If the *pause* argument is non-zero, then the specified volume or volume group will be set to paused state, and will remain in this state until either the volume is dismounted, or [PauseVolumeIndexer\(\)](#) is called again with *pause* set to zero. This method has no effect on virtual volumes, or on indexed volumes which have completed all indexing passes. Please also note that this is not available for iAP2 indexing.

3.28.21 BrowseMetadata

Read node metadata.

Syntax

```
CinemoError BrowseMetadata(  
    [in] const char * szid, /* node id */  
    [out] struct ICinemoMetapool ** pp /* return metapool */  
) ;
```

Parameters

- **szid**

The node identifier.

- **pp**

Return [ICinemoMetapool](#) containing node metadata.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

This method is implemented only for convenience. It is functionally equivalent to calling [ICinemoVFS::Open\(\)](#) on the MM server URL, specifying the node identifier and "browse=BrowseMetadata" parameters. It will return all metadata for the specified node.

3.28.22 BrowseDirectChildren

Read node child metadata.

Syntax

```
CinemoError BrowseDirectChildren(
    [in] const char * szid,                      /* node id */
    [out] struct ICinemoMetapool ** pp           /* return metapool */
);
```

Parameters

- **szid**

The node identifier.

- **pp**

Return [ICinemoMetapool](#) containing node child metadata.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

This method is implemented only for convenience. It is functionally equivalent to calling [ICinemoVFS::Open\(\)](#) on the MM server URL, specifying the node identifier and “browse=BrowseDirectChildren” parameters. It will return the metadata for all children of the specified node. Care should be taken to avoid calling this method on nodes with a large number of children, since the returned [ICinemoMetapool](#) will consume significant resources.

3.28.23 Search

Returns the result of a search.

Syntax

```
CinemoError Search(
    [in] const char * szid,                      /* node id */
    [in] const char * szexpr,                     /* upnp search expression */
    [in] const char * szsortcriteria,             /* upnp sort criteria */
    [out] struct ICinemoMetapool ** pp           /* return metapool */
);
```

Parameters

- **szid**
The node identifier.
- **szexpr**
The UPnP search expression.
- **szsortcriteria**
The UPnP sort criteria.
- **pp**
Return [ICinemoMetapool](#) containing search results.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

This method is implemented only for convenience. It is functionally equivalent to calling [ICinemoVFS::Open\(\)](#) on the MM server URL, specifying the node identifier, "search=<szexpr>" and "sort=<szsortcriteria>" parameters. It will return the metadata for all children of the specified node which meet the search criteria. Care should be taken to avoid calling this method on nodes with a large number of children, since the returned [ICinemoMetapool](#) will consume significant resources.

3.28.24 SetOption

Set configuration options for the given file identifier. This method supports setting of volume-specific options.

Syntax

```
CinemoError SetOption(
    [in] uint64 id                         /* volume id */
    [in] const char * szid,                  /* option id */
    [in] const char * szvalue                /* value */
);
```

Parameters

- **id**

Identifier of the volume.

- **szid**

Configuration option identifier.

- **szvalue**

Value to set.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may be called on a valid volume id to set volume-specific configuration options. The following volume-specific options are supported:

- *CINEMO_OPTION_MM_QUOTA_FOLDER_DEPTH*
- *CINEMO_OPTION_MM_QUOTA_FOLDERS_PER_VOLUME*
- *CINEMO_OPTION_MM_QUOTA_TRACKS_PER_FOLDER*
- *CINEMO_OPTION_MM_QUOTA_TRACKS_PER_VOLUME*
- *CINEMO_OPTION_MM_QUOTA_PLAYLISTS_PER_VOLUME*
- *CINEMO_OPTION_MM_QUOTA_PLAYLISTTRACKS_PER_VOLUME*
- *CINEMO_OPTION_MM_QUOTA_PLAYLISTTRACKS_PER_PLAYLIST*
- *CINEMO_OPTION_MM_FILENAME_WHITELIST*
- *CINEMO_OPTION_MM_FILENAME_EXCLUDE*
- *CINEMO_OPTION_MM_FILENAME_EXCLUDE_FULLPATH*
- *CINEMO_OPTION_MM_INDEXER_FASTPLAY*
- *CINEMO_OPTION_MM_INDEXER_FILESYSTEM_HINT*
- *CINEMO_OPTION_MM_INDEXER_SKIP_DOT_FILES*
- *CINEMO_OPTION_MM_INDEXER_SKIP_DOT_FOLDERS*
- *CINEMO_OPTION_MM_INDEXER_SKIP_HIDDEN_FILES*

- `CINEMO_OPTION_MM_INDEXER_SKIP_HIDDEN_FOLDERS`
- `CINEMO_OPTION_MM_INDEXER_MINIMUM_FILESIZE`
- `CINEMO_OPTION_MM_INDEXER_METADATA_SKIP`
- `CINEMO_OPTION_MM_INDEXER_COVERART_SKIP`
- `CINEMO_OPTION_MM_INDEXER_NOMEDIA_FOLDER_TAGS`
- `CINEMO_OPTION_MM_INDEXER_NOMEDIA_FILES_THRESHOLD`
- `CINEMO_OPTION_MM_INDEXER_CLASS_FROM_FILENAME`
- `CINEMO_OPTION_MM_THUMB_SKIP_VIDEOS`
- `CINEMO_OPTION_MM_THUMB_SKIP_IMAGES`

The volume-specific option is persisted immediately and applied on subsequent [MountVolume\(\)](#) or [ReindexVolume\(\)](#) calls.

3.28.25 GetOption

Get configuration options for the given file identifier. This method supports retrieving of volume-specific options.

Syntax

```
CinemoError GetOption(  
    [in]  uint64 id                      /* volume id */  
    [in]  const char * szid,                /* option id */  
    [out] struct ICinemoUTF8 ** pp        /* value */  
) ;
```

Parameters

- **id**

Identifier of the volume.

- **szid**

Configuration option identifier.

- **pp**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may be called on a valid volume id to get volume-specific configuration options. See [SetOption\(\)](#) for a list of available options. If the method succeeds, the pp will point to an [ICinemoUTF8](#) instance that will hold the option value as a string. This instance has to be manually released afterwards, using the Release method.

3.28.26 GetFieldMap

Get the Metaname field mapping table.

Syntax

```
CinemoError GetFieldMap(
    [out] const CinemoMMFieldMap ** ppmap           /* return field map */
);
```

Parameters

- **ppmap**

Return [CinemoMMFieldMap](#) containing the field map.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

This method returns a mapping table for metadata and their corresponding field ids, created from the `cinemo_mm.xml` file. The table can be used to lookup field names, e.g. for the ids returned in a *CinemoMMNodeEvent*.

The *CinemoMMFieldMap* is defined as follows:

```
typedef struct {
    const char * id;                      /* field id */
    CINEMO_METANAME metaname;             /* metaname */
} CinemoMMFieldMap;
```

3.28.27 GetHierarchyNodes

Get hierarchy containers for node.

Syntax

```
CinemoError GetHierarchyNodes (
    [in] const char * szid,                      /* node id */
    [out] struct ICinemoMetapool ** pp           /* return metapool */
);
```

Parameters

- **szid**

The node identifier.

- **pp**

Return [ICinemoMetapool](#) containing hierarchy node metadata.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method returns a metapool containing metadata of all hierarchy containers that a node is linked in and having *role* set in the media management configuration (see [Cinemo MM Configuration](#)). The role of the node will be included in the pool using the metaname *CINEMO_METANAME_VFS_UPNP_HIERARCHY_NODE_ROLE*.

3.28.28 GetParentNodes

Get parent containers for node.

Syntax

```
CinemoError GetParentNodes(
    [in] const char * szid,                      /* node id */
    [out] struct ICinemoMetapool ** pp           /* return metapool */
);
```

Parameters

- **szid**

The node identifier.

- **pp**

Return [ICinemoMetapool](#) containing parent node metadata.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method returns a metapool containing metadata of all parent containers from the volume browse.

3.28.29 StartVolumeEvents

Start receiving volume events.

Syntax

```
CinemoError StartVolumeEvents (
    [in] void * puser,                      /* user value */
    [in] CinemoMMVolumeEventCallback pcall   /* user callback function */
);
```

Parameters

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[StartVolumeEvents\(\)](#) will register an application defined callback function to receive volume events. Volume events are triggered when a volume is added, removed, mounted or dismounted. An application implementing the *CinemoMMVolumeEventCallback* function is able to know, at any point in time, the complete state of all volumes in the Cinemo MM server. This may be useful, for example, to manage Gracenote collection summaries.

The callback function is of the following form:

Syntax

```
CinemoError CinemoMMVolumeEventCallback (
    [in] void * puser,                      /* user data */
    [in] uint32 flags,                     /* flags */
    [in] const CinemoMMVolumeEvent * pevents, /* events */
    [in] int nevents,                      /* number of events */
    [in] CinemoError error_code           /* error code */
);
```

Parameters

- **puser**

User-defined parameter passed to [StartVolumeEvents\(\)](#).

- **puser**

Flags for this callback.

- **pevents**

Array of CinemoMMVolumeEvents.

- **nevents**

Number of entries in pevents.

- **error_code**

Error code in case of failure.

The *pevent* parameter contains an array of the CinemoMMVolumeEvent structure, which is defined as follows:

```
typedef struct {
    uint64 volume_id;                                /* volume id */
    CINEMO_MM_VOLUME_EVENT event_code;               /* event code */
    uint32 d[1];
    uint32 reserved[4];
} CinemoMMVolumeEvent;
```

The CINEMO_MM_VOLUME_EVENT field may take one of the following values:

- CINEMO_MM_VOLUME_EVENT_NONE
- CINEMO_MM_VOLUME_EVENT_EXISTING
- CINEMO_MM_VOLUME_EVENT_ADD
- CINEMO_MM_VOLUME_EVENT_REMOVE
- CINEMO_MM_VOLUME_EVENT_MOUNT
- CINEMO_MM_VOLUME_EVENT_DISMOUNT
- CINEMO_MM_VOLUME_EVENT_STATUS

On starting the volume events, the full state of volumes is signalled to the application in terms of a sequence of CINEMO_MM_VOLUME_EVENT_EXISTING events. For this initial callback, the *flags* parameter has the CINEMO_MM_VOLUME_EVENT_FLAG_FULLSTATE set. The application needs to compare this full state with its internal state. Note that a callback with CINEMO_MM_VOLUME_EVENT_FLAG_FULLSTATE set can be called with nevents equal to zero. This indicates that currently no volumes are handled by MM.

After the initial full state has been signalled, additions and removals of volumes are signalled by means of CINEMO_MM_VOLUME_EVENT_ADD and CINEMO_MM_VOLUME_EVENT_REMOVE events.

Changes in the mount state of the volume are signalled by means of the CINEMO_MM_VOLUME_EVENT_MOUNT and CINEMO_MM_VOLUME_EVENT_DISMOUNT events.

The status of the volume indexer is signalled using the CINEMO_MM_VOLUME_EVENT_STATUS event, where the d[0] parameter corresponds to the CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER bitmask.

The error code in the callback needs to be checked every time to verify no error has occurred. If you receive a callback with an error code other than *CinemoNoError*, the volume event thread encountered an error and had to be terminated. You will then no longer receive any further *CinemoMMVolumeEventCallback* callbacks.

HUMAX Confidential

3.28.30 StopVolumeEvents

Stop receiving volume events.

Syntax

```
CinemoError StopVolumeEvents();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[StopVolumeEvents\(\)](#) will unregister an application defined callback function previously registered with [StartVolumeEvents\(\)](#). After successful return from this method, the callback function will no longer be called. If no callback function was previously registered, the method will return *CinemoErrorState*.

3.28.31 StartNodeEvents

Start receiving node events for the specified volume.

Syntax

```
CinemoError StartNodeEvents(
    [in] const char * szclient,           /* client id */
    [in] uint64 volume_id,              /* volume id */
    [in] const char * szrevision,        /* last known revision */
    [in] void * puser,                  /* user value */
    [in] CinemoMMNodeEventCallback pcall); /* user callback function */
```

Parameters

- **szclient**

The client identifier.

- **volume_id**

The volume identifier.

- **szrevision**

The revision of the last received node event.

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[StartNodeEvents\(\)](#) will register an application defined callback function to receive node events for a specified volume. Node events are triggered when a node is added to a volume, removed from a volume, or edited. An application implementing the *CinemoMMNodeEventCallback* function is able to know, at any point in time, the complete state of all nodes within an MM volume. This may be useful, for example, to manage Gracenote collection summaries.

The *szclient* parameter identifies the application. The identifier needs to be unique among applications using the node event API.

The *szrevision* parameter is an opaque revision identifier of the last received node event. The application defined *CinemoMMNodeEventCallback* function will be called with a unique *szrevision* parameter for each node event.

The application should not attempt to interpret this parameter. If an application wishes to stop and restart receiving node events, it should remember the last received *szrevision*, and pass the unmodified parameter to [StartNodeEvents\(\)](#) in order to resume events from the last known revision. If the last known revision is unknown, a NULL revision value should be passed.

The callback function is of the following form:

Syntax

```
CinemoError CinemoMMNodeEventCallback(
    [in] void * puser,                                /* user data */
    [in] uint64 volume_id,                            /* volume id of the node */
    [in] const CinemoMMNodeEvent * pevents,           /* events */
    [in] int nevents,                                /* number of events */
    [in] const char * szrevision,                     /* revision */
    [in] CinemoError error_code                      /* error code */
);
```

Parameters

- **puser**

User-defined parameter passed to [StartNodeEvents\(\)](#).

- **volume_id**

Volume identifier for the node.

- **pevents**

Array of CinemoMMNodeEvents.

- **nevents**

Number of entries in pevents.

- **szrevision**

Revision identifier for this event.

- **error_code**

Error code in case of failure.

For each node event a separate CinemoMMNodeEventCallback is called. It contains an array of the CinemoMMNodeEvent structure, which is defined as follows:

```
typedef struct {
    uint64 node_id;                                /* node id */
    CINEMO_MM_NODE_EVENT event_code;                /* event code */
    const uchar * fields;                           /* zero terminated list of fields changed */
    const struct ICinemoMetapool * ppool;          /* node metapool */
    uint32 d[1];
    uint32 reserved[2];
} CinemoMMNodeEvent;
```

All metadata fields, which have changed since the last known revision, will be encoded in the fields variable. A mapping table, for retrieving metanames from field ids can be created with the [ICinemoMM::GetFieldMap\(\)](#) function. The new metadata items are contained in the node metapool.

The CINEMO_MM_NODE_EVENT field may take one of the following values:

- CINEMO_MM_NODE_EVENT_NONE
- CINEMO_MM_NODE_EVENT_EXISTING
- CINEMO_MM_NODE_EVENT_ADD
- CINEMO_MM_NODE_EVENT_REMOVE
- CINEMO_MM_NODE_EVENT_CHANGE
- CINEMO_MM_NODE_EVENT_STATUS

On initial start of the node events with revision NULL or 0, a full state will be signalled by means of a sequence of CINEMO_MM_NODE_EVENT_EXISTING events. For all subsequent revisions, changes are signalled by means of CINEMO_MM_NODE_EVENT_ADD, CINEMO_MM_NODE_EVENT_REMOVE, and CINEMO_MM_NODE_EVENT_CHANGE events. The CINEMO_MM_NODE_EVENT_STATUS informs the application about the current indexing status of the volume. The status can be retrieved from the d[0] parameter and corresponds to the CINEMO_MM_VOLUME_STATUS_FLAG_INDEXER bitmask.

By default, the pool associated to each event will be NULL. The application can define which metadata shall be forwarded in the node event pool by setting URL parameters in the requested revision. This is achieved by using the API call [ICinemoUTF8::SetURLParameter\(\)](#) on the szrevision passed to [StartNodeEvents\(\)](#). The following parameters (and combinations of them) are available:

- *metapool=1*
The full set of metadata will be forwarded on ADD and EXISTING events, the delta will be forwarded on CHANGE events.
- *metadata=field_id1,field_id2,...*
Comma separated list of metadata fields that shall be forwarded for any ADD, CHANGE, or EXISTING event.

Note that the pool provided by the node event corresponds to the most recent metadata held by MM. For already removed nodes, this pool can be NULL, even with the above parameters set.

The error code in the callback needs to be checked every time to verify no error has occurred. If you receive a callback with an error code other than *CinemoNoError* or *CinemoErrorRange*, the volume event thread encountered an error and had to be terminated. You will then no longer receive any further *CinemoMMVolumeEventCallback* callbacks.

A *CinemoErrorRange* indicates that the application used a revision that is too old. In order to recover from this error, all persistent information on the volume should be discarded and *CinemoNoError* should be returned from the callback in order to restart the events from revision NULL. Note that a *CinemoErrorRange* can only occur if the node journal is limited in size (see *CINEMO_OPTION_MM_JOURNAL_NODE_EVENT_LIMIT*).

3.28.32 StopNodeEvents

Stop receiving node events for the specified volume.

Syntax

```
CinemoError StopNodeEvents(  
    [in]  uint64 volume_id  
);  
/* volume id */
```

Parameters

- **volume_id**

The volume identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[StopVolumeEvents\(\)](#) will unregister an application defined callback function previously registered with [StartNodeEvents\(\)](#). After successful return from this method, the callback function will no longer be called. If no callback function was previously registered for the specified volume, the method will return *CinemoErrorState*.

3.28.33 StartIndexingEvents

Start receiving indexing events for the specified volume.

Syntax

```
CinemoError StartIndexingEvents(
    [in] uint64 volume_id,                      /* volume id */
    [in] void * puser,                           /* user value */
    [in] CinemoMMIndexingEventCallback pcall    /* user callback function */
);
```

Parameters

- **volume_id**

The volume identifier.

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[StartIndexingEvents\(\)](#) will register an application defined callback function to receive indexing events for a specified indexed volume. Indexing events are triggered when an indexed volume is mounted and its tracks are indexed for the first time. An application implementing the *CinemoMMIndexingEventCallback* function is able to modify the metadata of a node which has just been indexed. This may be used, for example, to trigger Gracenote recognition algorithms or perform other kinds of metadata normalization.

The callback function is of the following form:

Syntax

```
CinemoError CinemoMMIndexingEventCallback(
    [in] void * puser,                          /* user data */
    [in] uint64 volume_id,                     /* volume id of the node */
    [in] const CinemoMMIndexingEvent * pevents, /* events */
    [in] int nevents,                         /* number of events */
    [in] CinemoError error_code               /* error code */
);
```

Parameters

- **puser**

User-defined parameter passed to [StartIndexingEvents\(\)](#).

- **volume_id**

Volume identifier for the node.

- **pevents**

Array of CinemoMMIndexingEvents.

- **nevents**

Number of entries in pevents.

- **szrevision**

Revision identifier for this event.

- **error_code**

Error code in case of failure.

For each node event a separate CinemoMMIndexingEventCallback is called. It contains an array of the CinemoMMIndexingEvent structure, which is defined as follows:

```
typedef struct {
    uint64 node_id;                                /* node id */
    uint32 indexing_pass_id;                         /* pass id */
    uint32 flags;                                   /* CINEMO_MM_INDEXING_EVENT_FLAG bitmask */
    const struct ICinemoMetapool * ppool;           /* node metapool */
    uint32 reserved[3];
} CinemoMMIndexingEvent;
```

The following bits can be set on the flags bitmask:

- CINEMO_MM_INDEXING_EVENT_FLAG_NONE
- CINEMO_MM_INDEXING_EVENT_FLAG_FIRST
- CINEMO_MM_INDEXING_EVENT_FLAG_LAST

The event flags CINEMO_MM_INDEXING_EVENT_FLAG_FIRST and FLAG_LAST indicate the beginning and end of each burst of events. In particular, the first node in each indexing pass will have the CINEMO_MM_INDEXING_EVENT_FLAG_FIRST flag set, the last node will have the CINEMO_MM_INDEXING_EVENT_FLAG_LAST flag set.

The error code in the callback needs to be checked every time to verify no error has occurred. If you receive a callback with an error code other than *CinemoNoError*, the volume event thread encountered an error and had to be terminated. You will then no longer receive any further CinemoMMVolumeEventCallback callbacks.

3.28.34 StopIndexingEvents

Stop receiving indexing events for the specified volume.

Syntax

```
CinemoError StopIndexingEvents(  
    [in]  uint64 volume_id, /* volume id */  
) ;
```

Parameters

- **volume_id**

The volume identifier.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[StopIndexingEvents\(\)](#) will unregister an application defined callback function previously registered with [StartIndexingEvents\(\)](#). After successful return from this method, the callback function will no longer be called. If no callback function was previously registered for the specified volume and indexing pass, the method will return *CinemoErrorState*.

3.28.35 RegisterQueryHandler

Register a query expression handler.

Syntax

```
CinemoError RegisterQueryHandler(
    [in] const char * szhandler,           /* query handler */
    [in] void * puser,                   /* user value */
    [in] CinemoMMQueryHandler pcall     /* user callback function */
);
```

Parameters

- **szhandler**

The query handler.

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[RegisterQueryHandler\(\)](#) method will register an application defined callback function to receive application specific queries. An application specific query is triggered when an unknown query expression is received by the Cinemo MM server. For example, the following code will trigger a query for the handler "gracenote_search":

```
ICinemoPlaylist::Open("upnp://xxx.xxx.xxx.xxx:xxxx/0/MediaServer/DeviceDesc.xml"
    "?pid=0"
    "&search=gracenote_search$<expr>");
```

The *<expr>* parameter may be any text which can be URL encoded. Cinemo MM will not attempt to interpret the expression, and will pass the unmodified text to the *CinemoMMQueryHandler* registered for the specific handler. The application should return query results as a list of node identifiers, one for each track. The node identifiers must reference valid nodes in the MM node hierarchy. Invalid nodes will be stripped from the result set.

The callback function is of the following form:

Syntax

```
CinemoError CinemoMMQueryHandler(  
    [in] void * puser, /* user data */  
    [in] const char * szhandler, /* handler */  
    [in] const char * szargs, /* arguments */  
    [in] uint64 node_id, /* node id */  
    [out] struct ICinemoBlob ** pp /* return query results as array of node ids */  
) ;
```

Parameters

- **puser**

User-defined parameter passed to [RegisterQueryHandler\(\)](#).

- **szhandler**

Identification string for the query handler.

- **szargs**

Arguments to the search query.

- **node_id**

Node id of the container to search.

- **pp**

ICinemoBlob to return the result.

For each search query a separate CinemoMMQueryHandler is called. It contains the identifier for the handler, registered with [RegisterQueryHandler\(\)](#), as well as the search query as parameters. The *node_id* is the id of the container to execute the search upon, specified as *pid* in the query.

The result of the query has to be returned as an ICinemoBlob, which contains an array of uint64 node_ids.

3.28.36 UnregisterQueryHandler

Unregister a query handler.

Syntax

```
CinemoError UnregisterQueryHandler(  
    [in] const char * szhandler /* query handler */  
) ;
```

Parameters

- **szhandler**

The query handler.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[UnregisterQueryHandler\(\)](#) method will unregister an application defined callback function previously registered with [RegisterQueryHandler\(\)](#). After successful return from this method, the callback function will no longer be called. If no callback function was previously registered for the specified query handler, the method will return *CinemoErrorState*.

3.28.37 RegisterMetadataHandler

Register a metadata expression handler.

Syntax

```
CinemoError RegisterMetadataHandler(
    [in] const char * szhandler,           /* metadata handler */
    [in] void * puser,                   /* user value */
    [in] CinemoMMMetadataHandler pcall   /* user callback function */
);
```

Parameters

- **szhandler**

The metadata handler.

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[RegisterMetadataHandler\(\)](#) method will register an application defined callback function to receive application specific metadata queries. An application specific metadata query is triggered when an unknown query expression is received by the Cinemo MM server. For example, the following code will trigger a metadata query for the handler "gracenote_metadata":

```
ICinemoPlaylist::Open("upnp://xxx.xxx.xxx.xxx:xxxx/0/MediaServer/DeviceDesc.xml"
    "?pid=gracenote_metadata$<expr>"
    "&browse=BrowseMetadata");
```

The *<expr>* parameter may be any text which can be URL encoded. Cinemo MM will not attempt to interpret the expression, and will pass the unmodified text to the *CinemoMetadataHandler* registered for the specific handler. The application should return metadata query results as a single [CinemoMetapool](#).

The callback function is of the following form:

Syntax

```
CinemoError CinemoMetadataHandler(
```

```
[in] void * puser,          /* user data */
[in] const char * szhandler, /* handler */
[in] const char * szargs,   /* arguments */
[out] struct ICinemoMetapool ** pp    /* return metapool */
);
```

Parameters

- **puser**

User-defined parameter passed to [RegisterMetadataHandler\(\)](#).

- **szhandler**

Identification string for the handler.

- **szargs**

Arguments to the search query.

- **pp**

Metapool for returning the query results.

For each metadata search query a separate CinemoMetadataHandler is called. It contains the identifier for the handler registered with [RegisterMetadataHandler\(\)](#) and the search query as parameters.

3.28.38 UnregisterMetadataHandler

Unregister a metadata request handler.

Syntax

```
CinemoError UnregisterMetadataHandler(  
    [in] const char * szhandler /* metadata request handler */  
) ;
```

Parameters

- **szhandler**

The metadata request handler.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

[UnregisterQueryHandler\(\)](#) method will unregister an application defined callback function previously registered with [RegisterQueryHandler\(\)](#). After successful return from this method, the callback function will no longer be called. If no callback function was previously registered for the specified query handler, the method will return *CinemoErrorState*.

3.28.39 SetCachedRevision

This method stores a journaling revision in the persistent volume database.

Syntax

```
CinemoError SetCachedRevision (
    [in] const char * szapp                                /* application id */
    [in] uint64 volume_id                                 /* volume id */
    [in] const char * szrevision                          /* revision */
);
```

Parameters

- **szapp**

Identifier for the application storing the revision.

- **volume_id**

Volume identifier.

- **szrevision**

Node event revision to store.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Please note that node event revision should be stored atomically in the external database that is synchronized using node events. If this method is used to store node events, the application needs to handle possible inconsistencies between stored revision and external database.

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

3.28.40 GetCachedRevision

This method allows to retrieve a journaling revision from the persistent volume database.

Syntax

```
CinemoError GetCachedRevision (
    [in] const char * szapp                                /* application id */
    [in] uint64 volume_id                                 /* volume id */
    [out] struct ICinemoUTF8 ** pprevision               /* revision */
);
```

Parameters

- **szapp**

Identifier for the application storing the revision.

- **volume_id**

Volume identifier.

- **pprevision**

The address of a pointer to receive the requested [ICinemoUTF8](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, it will return an [ICinemoUTF8](#) interface that holds the revision string. This interface must be released when no longer required, using the [ICinemoUnknown::Release\(\)](#) method.

This method may only be called after a connection to the MM server has been established by either [CreateServer\(\)](#) or [ConnectToServer\(\)](#).

3.29 ICinemoEventQueue

The [ICinemoEventQueue](#) interface provides a simple thread-safe FIFO for events.

ICinemoEventQueue Methods

METHOD	DESCRIPTION
<u>Read</u>	Read an event from the queue
<u>Post</u>	Post an event to the queue
<u>ReadCancel</u>	Cancel any Read calls
<u>ReadEnable</u>	Enable any Read calls after a Cancel
<u>RemoveAll</u>	Remove all events from queue

Notes

This interface allows the calling application to receive events via a thread-safe method. Events can be received immediately (if available), or a timeout can be set to wait for an event to arrive. If a timed or infinite read needs to be cancelled and to disable future Read calls, the ReadCancel method can be called to effectively disable the queue and cancel any pending Read calls. Similarly, to re-enable the read, the ReadEnable method can be called to allow future calls to Read to return events.

3.29.1 Read

Read an event from the queue.

Syntax

```
CinemoError Read(
    [out]    CinemoEvent& e,
    [in]     uint32 timeout_ms
);
```

Parameters

- **e**
Event buffer to return event from queue.
- **timeout_ms**
Timeout value in milliseconds for the read.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns *CinemoErrorTimeout* (based on the value of `timeout_ms`), *CinemoErrorCancel* (if a `ReadCancel` is called), or another *CinemoError* code.

Remarks

This method implements a blocking read, and returns either an event or an error code. The `timeout_ms` parameter specifies how many milliseconds to wait before returning *CinemoErrorTimeout*. If set to zero, function will return immediately, with or without an event. If set to `UINT32_MAX`, the timeout is infinite.

3.29.2 Post

Post an event to the queue. If queue is full, function will return immediately with `CinemoErrorOverflow`.

Syntax

```
CinemoError Post(  
    [in]      const CinemoEvent& e  
) ;
```

Parameters

- **e**

Event to post to queue.

Return value

If the method succeeds, it returns `CinemoNoError`. Otherwise, it returns a `CinemoError` code.

Remarks

The successful return of this method simply depends on the successful insertion of the event into the FIFO.

3.29.3 ReadCancel

Cancel any Read calls.

Syntax

```
CinemoError ReadCancel();
```

Return value

The method returns *CinemoNoError*.

Remarks

This method will unblock any current [Read\(\)](#) call. All subsequent calls to [Read\(\)](#) will return *CinemoErrorCancel*.

3.29.4 ReadEnable

Enable any Read calls after a Cancel.

Syntax

```
CinemoError ReadEnable();
```

Return value

The method returns *CinemoNoError*.

Remarks

This method will re-enable the queue after calling ReadCancel.

3.29.5 RemoveAll

Remove all events from queue.

Syntax

```
CinemoError RemoveAll();
```

Return value

The method returns *CinemoNoError*.

3.30 ICinemoMuxer

The [ICinemoMuxer](#) interface provides simple multiplexing capability. Please note that this interface is not thread-safe.

ICinemoMuxer Methods

METHOD	DESCRIPTION
<u>Initialize</u>	Initialize the multiplexer
<u>Finalize</u>	Complete the multiplex
<u>AddTrack</u>	Add a new track to the target multiplex
<u>DeliverSample</u>	Deliver a single multiplex sample

Notes

This interface allows the calling application to multiplex tracks.

3.30.1 Initialize

Initialize the multiplexer.

Syntax

```
CinemoError Initialize(
    [in] CINEMO_MUXFORMAT format,           /* multiplex target format */
    [in] ICinemoFile * pfile,               /* target file */
    [in] ICinemoMetapool * ppool,           /* optional (for writing tags) */
    [in] const char * szparam,              /* optional (additional parameters) */
);

```

Parameters

- **format**

Desired target format for the multiplex (only MP4 is supported at this time).

- **pfile**

Target ICinemoFile object.

- **ppool**

Optional metapool for writing tags.

- **szparam**

Optional parameters specified in the format of URL parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method initializes the [ICinemoMuxer](#), including setting the format for the multiplex, and optionally a [ICinemoMetapool](#) to add metadata and user data for the I/O callback function. The metadata for the multiplex needs to be available in the specified metadata pool latest when [ICinemoMuxer::Finalize\(\)](#) is called. The following metadata attributes are supported for the ISO/MP4 format (numeric values may also be represented as string):

- Title
- Composer
- Album
- Year
- Comment
- Genre
- Artist
- Description
- iTunGapless

- DiscNumber
- TrackNumber
- Image
- Keyword
- BeatsPerMinute
- AlbumArtist
- Compilation

For compatibility with existing applications which use the legacy callback approach for writing the multiplex output, there is an adapter object available which can be created with [CinemoCreateMuxerLegacyFile\(\)](#). Related sample code is available in SampleAudioCodec.

3.30.2 Finalize

Complete the multiplex.

Syntax

```
CinemoError Finalize();
```

Return value

This method returns *CinemoNoError*.

Remarks

This method releases all interfaces associated with the [ICinemoMuxer](#).

3.30.3 AddTrack

Add a new track to the target multiplex.

Syntax

```
CinemoError AddTrack(  
    [in]     const CinemoMediaType& type,      /* type of track */  
    [out]    uint32& track_id                  /* return track id */  
) ;
```

Parameters

- **type**

Type of track to add.

- **track_id**

Track ID returned after track is added.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method adds a track to the multiplex. If the track has been successfully added, the Track ID is returned in the *track_id* variable.

3.30.4 DeliverSample

Deliver a single multiplex sample.

Syntax

```
CinemoError DeliverSample(
    [in]      uint32 track_id,                      /* as returned by AddTrack() */
    [in]      const CinemoMuxSampleAttributes& attrs
);
```

Parameters

- **track_id**

Track ID of the desired sample.

- **attrs**

Attributes of the sample.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method delivers a sample from the muxer, given the Track ID and the attributes of the sample.

3.31 ICinemoDDP

The [ICinemoDDP](#) interface provides access to a Cinemo DDP server. It can be created using the *CinemoCreateDDP()* function defined in *cinemo.h*.

This interface may be used to create a local Cinemo DDP server. Please note that this interface is not thread-safe.

ICinemoDDP Methods

METHOD	DESCRIPTION
<u>CreateServer</u>	Create a Cinemo DDP server
<u>GetServerURL</u>	Returns the Cinemo DDP server URL.

3.31.1 CreateServer

Create a Cinemo DDP server.

Syntax

```
CinemoError CreateServer(  
    [in] const char * szurl  
) ;
```

Parameters

- **szurl**

Specifies the Cinemo DDP server URL (ddp://xxx).

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

On successful return, a new instance of a Cinemo DDP server will be instantiated and available on the network. The server will remain active until the [ICinemoDDP](#) interface is released, after which all its resources will be freed.

3.31.2 GetServerURL

Returns the Cinemo DDP server URL.

Syntax

```
CinemoError GetServerURL(  
    [out] struct ICinemoUTF8 ** pp  
) ;
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The returned URL will be the localhost address and port number of the server created by [CreateServer\(\)](#).

3.32 ICinemoWindowBitmap

The [ICinemoWindowBitmap](#) interface provides capabilities in a thread-safe manner to load, resize and store images in various formats. It can be created using the *CinemoCreateBitmap()* function defined in *cinemo.h*.

ICinemoWindowBitmap Methods

METHOD	DESCRIPTION
<u>Assign</u>	Assign frame.
<u>AssignUpdate</u>	Assign frame with update region hint.
<u>AssignImage</u>	Assign image.
<u>SaveImage</u>	Save image.
<u>Resize</u>	Resize bitmap.
<u>GetSize</u>	Get size.
<u>Premultiply</u>	Premultiply RGB channels.
<u>Lock</u>	Get direct buffer access.
<u>Unlock</u>	Release direct buffer access.
<u>DrawRect</u>	Draw rectangle with color.
<u>DrawString</u>	Draw basic string.
<u>DrawBitmap</u>	Draw bitmap.

3.32.1 Assign

Assigns a uncompressed frame to the bitmap.

Syntax

```
CinemoError Assign(  
    [in]  const CinemoFrame& frame  
) ;
```

Parameters

- **frame**

Specifies the uncompressed frame to be loaded.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.2 AssignUpdate

Assigns a uncompressed frame to the bitmap with update region hint.

Syntax

```
CinemoError AssignUpdate(  
    [in] const CinemoFrame& frame,  
    [in] const CinemoRect * pregion,  
    [in] uint32 nregion  
) ;
```

Parameters

- **frame**

Specifies the uncompressed frame to be loaded.

- **pregion**

Specifies list of changed rectangles as update region hint.

- **nregion**

Specifies number of changed rectangles.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.3 AssignImage

Assigns a compressed image to the bitmap.

Syntax

```
CinemoError AssignImage(  
    [in] const void * pdata,  
    [in] uint32 nbytes  
) ;
```

Parameters

- **pdata**
Pointer to the image data to be loaded.
- **nbytes**
Size of the image data in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.4 SaveImage

Saves bitmap as compressed image.

Syntax

```
CinemoError SaveImage (
    [out] struct ICinemoBlob ** pp,
    [in] const CinemoImageFormat& format
);
```

Parameters

- **pp**

The address of a pointer to receive the requested ICinemoBlob interface.

- **format**

Output format definition.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a CinemoError code.

3.32.5 Resize

Resize bitmap buffer. Previous content is lost.

Syntax

```
CinemoError Resize(  
    [in] uint32 cx,  
    [in] uint32 cy  
) ;
```

Parameters

- **cx**

Width in pixels.

- **cy**

Height in pixels.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.6 GetSize

Get height and width of bitmap.

Syntax

```
CinemoError GetSize(  
    [out] CinemoSize& size  
) ;
```

Parameters

- **size**

The size of the bitmap.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Please note that the returned size will be 0/0, if no image has been assigned before calling this API.

3.32.7 Premultiply

Premultiply RGB channels with alpha channel.

Syntax

```
CinemoError Premultiply();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.8 Lock

Get direct access to bitmap buffer.

Syntax

```
CinemoError Lock(  
    [out] CinemoFrame& frame  
) ;
```

Parameters

- **frame**

Buffer attributes and pointer returned for direct access.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.9 Unlock

Release direct bitmap buffer access.

Syntax

```
CinemoError Unlock();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.10 DrawRect

Draw rectangle with solid color.

Syntax

```
CinemoError DrawRect(  
    [in] const CinemoRect& rc,  
    [in] uint32 ncolor  
) ;
```

Parameters

- **rc**
Rectangle area.
- **ncolor**
ARGB color value to draw rectangle.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.11 DrawString

Draw basic string on bitmap.

Syntax

```
CinemoError DrawString(  
    [in] const CinemoRect& rc,  
    [in] const CinemoFontStyle& style,  
    [in] const char * szchars  
) ;
```

Parameters

- **rc**

Rectangle area.

- **style**

Font painting and alignment attributes.

- **szchars**

Pointer to a zero terminated UTF-8 character string.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.32.12 DrawBitmap

Draw and stretch bitmap into rectangle area.

Syntax

```
CinemoError DrawRect(  
    [in] const CinemoRect& rc,  
    [in] struct ICinemoWindowBitmap * pbitmap,  
    [in] uint32 flags  
) ;
```

Parameters

- **rc**

Rectangle area.

- **pbitmap**

Specifies the bitmap to draw.

- **flags**

Flags to control drawing.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.33 ICinemoWindowOverlay

[ICinemoWindowOverlay](#) is derived from [ICinemoWindowBitmap](#).

ICinemoWindowOverlay Methods

METHOD	DESCRIPTION
<u>SetVisible</u>	Show or hide the overlay
<u>SetCropping</u>	Set source rectangle for cropping
<u>SetScaling</u>	Set destination rectangle for scaling
<u>Sync</u>	Commit changes and repaint overlay

3.33.1 SetVisible

Show or hide the overlay.

Syntax

```
CinemoError SetVisible(  
    [in]      uint32 show
```

Parameters

- **show**

Flag indicating whether to show or hide the overlay.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

To commit this change it's necessary to call [Sync\(\)](#).

3.33.2 SetCropping

Set source rectangle for cropping.

Syntax

```
CinemoError SetCropping(  
    [in]      const CinemoRect &rcsrc  
) ;
```

Parameters

- **rcsrc**

Source rectangle for cropping.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

To commit this change it's necessary to call [Sync\(\)](#).

3.33.3 SetScaling

Set destination rectangle for scaling.

Syntax

```
CinemoError SetScaling(  
    [in]      const CinemoRect &rcdst  
) ;
```

Parameters

- **rcdst**

Destination rectangle for scaling.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

To commit this change it's necessary to call [Sync\(\)](#).

3.33.4 Sync

Commit changes and repaint overlay.

Syntax

```
CinemoError Sync();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.34 ICinemoWindow

The ICinemoWindow class should only be used in a production system after consultation with Cinemo.

ICinemoWindow Methods

METHOD	DESCRIPTION
<u>SetEventCallback</u>	Set the callback function to receive events
<u>SetEventQueue</u>	Set the event queue to receive events
<u>SetEventSourceID</u>	Set source ID for posted events
<u>SetWindowPos</u>	Set the window position
<u>SetScreenSaver</u>	Enable or disable screen saver mode
<u>SetFullscreen</u>	Enable or disable fullscreen mode
<u>SetPaintFreeze</u>	Temporarily disable painting
<u>GetDeviceName</u>	Get device_name in video parameters to use this window
<u>GetWindowDeviceName</u>	Get the window device name string with platform specific window and surface handles as URL parameters
<u>GetClientRect</u>	Get window client rectangle
<u>GetCaps</u>	Get video window driver capabilities
<u>InitBitmap</u>	Create a bitmap for off-screen drawing
<u>InitOverlay</u>	Create an overlay for on screen display

3.34.1 SetEventCallback

Set the callback function to receive events.

Syntax

```
CinemoError SetEventCallback(  
    [in]    void * puser,  
    [in]    CinemoEventCallback pcall  
) ;
```

Parameters

- **puser**

User-defined parameter passed to the application callback function.

- **pcall**

Application callback function.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The user defined application callback function will be called asynchronously whenever a new window event is available. The application is responsible for synchronizing events from multiple threads to its own application thread ([ICinemoEventQueue](#) interface may be used for this purpose).

To stop receiving events, a NULL callback function may be specified.

3.34.2 SetEventQueue

Set the event queue to receive events.

Syntax

```
virtual CinemoError SetEventQueue(  
    [in]     struct ICinemoEventQueue * pqueue  
) ;
```

Parameters

- **pqueue**

Pointer to [ICinemoEventQueue](#) object to receive events.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This approach helps transferring the events to the application main thread using an event queue object. It behaves the same as if an application callback would push the new events to the event queue. The queue object will be kept referenced by the window until it is set again or the window is destroyed.

To stop associating a queue with the window, a NULL pointer may be specified.

3.34.3 SetEventSourceID

Set source ID for posted events.

Syntax

```
CinemoError SetEventSourceID(  
    [in]      uint32 source_id  
) ;
```

Parameters

- **source_id**

User-defined number passed in *CinemoEvent* *source_id* member.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This approach helps differentiate sources of events when using shared event queue object.

3.34.4 SetWindowPos

Set the window position.

Syntax

```
CinemoError SetWindowPos(  
    [in]      const CinemoRect &rc  
) ;
```

Parameters

- **rc**

Rectangle specifying the window position.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Not all platforms support this feature.

3.34.5 SetScreenSaver

Enable or disable screen saver mode.

Syntax

```
CinemoError SetScreenSaver(  
    [in]      uint32 enable  
) ;
```

Parameters

- **enable**

Flag indicating if screen saver mode should be enabled.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Not all platforms support this feature.

3.34.6 SetFullscreen

Enable or disable fullscreen mode.

Syntax

```
CinemoError SetFullscreen(  
    [in]      uint32 enable  
) ;
```

Parameters

- **enable**

Flag indicating if fullscreen mode should be enabled.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Not all platforms support this feature.

3.34.7 SetPaintFreeze

Temporarily disable painting.

Syntax

```
CinemoError SetPaintFreeze(  
    [in]      uint32 enable  
) ;
```

Parameters

- **enable**

Flag indicating if paint freeze should be enabled.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Not all platforms support this feature, example where layer compositing is done natively by windowing system, like QNX Screen.

3.34.8 GetDeviceName

Get device_name in video parameters to use this window.

Syntax

```
CinemoError GetDeviceName (
    [in/out] CinemoVideoParams& params
);
```

Parameters

- **params**

Video device parameters structure reference.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method takes a *CinemoVideoParams* structure and updates its *device_name* to use this window.

3.34.9 GetWindowDeviceName

Get the window device name string with platform specific window and surface handles as URL parameters.

Syntax

```
CinemoError GetWindowDeviceName (
    [out]    struct ICinemoUTF8 ** pp
);
```

Parameters

- **pp**

The address of an ICinemoUTF8 pointer that will contain the window device name.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.34.10 GetClientRect

Get window client rectangle.

Syntax

```
CinemoError GetClientRect(  
    [out]    CinemoRect &rc  
) ;
```

Parameters

- **rc**

Returned rectangle.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.34.11 GetCaps

Get video window driver capabilities.

Syntax

```
uint32 GetCaps();
```

Return value

The return value is a bitmask of *CINEMO_WINDOW_CAPS* enumerated type.

3.34.12 InitBitmap

Create a bitmap for off-screen drawing.

Syntax

```
CinemoError InitBitmap(  
    [out]    struct ICinemoWindowBitmap ** pp  
) ;
```

Parameters

- **pp**

The address of an ICinemoWindowBitmap pointer that will contain the bitmap.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.34.13 InitOverlay

Create an overlay for on screen display.

Syntax

```
CinemoError InitOverlay(
    [in]     sint32 zorder,
    [in]     uint32 flags,
    [out]    struct ICinemoWindowOverlay ** pp
);
```

Parameters

- **zorder**

Specify the z-order number for the new overlay

- **flags**

CINEMO_OVERLAY_FLAGS bitmask.

- **pp**

The address of an ICinemoWindowOverlay pointer that will contain the overlay.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a CinemoError code.

3.35 ICinemoMediaType

The ICinemoMediaType class provides access to CinemoMediaType and the respective CinemoMediaTypeHeader

ICinemoMediaType Methods

METHOD	DESCRIPTION
<u>Serialize</u>	Create a platform independent binary representation of the instance
<u>Deserialize</u>	Initialize the instance based on the given binary representation
<u>Assign</u>	Initialize the instance based on the given CinemoMediaType
<u>Get</u>	Read only access to the CinemoMediaType

3.35.1 Serialize

Create a platform independent binary representation of the instance.

Syntax

```
CinemoError Serialize(  
    [out]    struct ICinemoBlob ** pp  
) const;
```

Parameters

- **pp**

The address of a pointer to receive the requested [ICinemoBlob](#) interface.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the method succeeds, it will return an [ICinemoBlob](#) interface that holds the binary representation. This interface must be released when no longer required, using the [ICinemoUnknown::Release\(\)](#) method.

3.35.2 Deserialize

Initialize this instance based on the given binary representation.

Syntax

```
virtual CinemoError Deserialize(  
    [in]    const void * pdata,  
    [in]    uint32 nbytes  
) ;
```

Parameters

- **pdata**

Pointer to the binary data.

- **nbytes**

Number of bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.35.3 Assign

Initialize the instance based on the given CinemoMediaType.

Syntax

```
virtual CinemoError Assign(  
    [in]      const CinemoMediaType& media  
);
```

Parameters

- **media**

Mediatype that is used to initialize the instance.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

3.35.4 Get

Read only access to the CinemoMediaType.

Syntax

```
virtual const CinemoMediaType& Get();
```

Return value

The method returns the reference to the underlying CinemoMediaType.

Remarks

The reference to the underlying CinemoMediaType is valid and constant as long as no other method of the instance is called.

4. Cinemo Interfaces for Android Auto

All Cinemo interfaces are defined in C++ header files which are provided with the Cinemo Media Engine SDK. These headers are pre-processed based on the enabled feature set for a project, and so may not include all the interfaces and API functions documented here. For example, [ICinemoMM](#) interface is only available when Cinemo Media Management feature is enabled. If you require an interface or function which is described in this document but is missing from the Cinemo SDK headers, then please contact your Cinemo representative to provide you an updated SDK with these features enabled.

All Cinemo interfaces are defined in C++ and follow the Component Object Model (COM) specification. As defined by COM, all Cinemo interfaces derive from a standard [ICinemoUnknown](#) interface, which consists of three methods:

- [ICinemoUnknown::AddRef\(\)](#) and [ICinemoUnknown::Release\(\)](#), reference counting to control the lifetime of interfaces.
- [ICinemoUnknown::QueryInterface\(\)](#), access to interfaces by ID.

All Cinemo interfaces are reflexive, symmetric, and transitive. The reflexive property means that calling [ICinemoUnknown::QueryInterface\(\)](#) on a given interface with the interface's ID will return the same instance of the interface. The symmetric property means that when interface B is retrieved from interface A by calling [ICinemoUnknown::QueryInterface\(\)](#), interface A is also retrievable from interface B. The transitive property means that if interface B is obtainable from interface A and interface C is obtainable from interface B, then interface C is also retrievable from interface A.

4.1 General remarks for Android Auto

The Android Auto receiver library provided by Google allows for certain endpoints to be registered with the Android device if they are required by the HU. Cinemo will try to detect if a certain endpoint needs to be registered, based on certain settings, which will be described below.

4.1.1 Endpoints

Audio source endpoint

The audio source endpoint will always be registered.

Audio sink endpoints

There are two audio channels available for Android Auto which will always be registered:

- Guidance (driver guidance / ASR and turn-by-turn)
- Media (cabin media)

Bluetooth endpoint

The bluetooth endpoint will be registered if an [ICinemoAndroidAutoBluetoothCallbacks](#) interface is provided and the `CINEMO_METANAME_ANDROIDAUTO_BLUETOOTH_ADDRESS` and `CINEMO_METANAME_ANDROIDAUTO_BLUETOOTH_PAIRING` metapool entries are present.

Input endpoint

A touchscreen will be registered for a display if all of the following metapool entries are provided:

- `CINEMO_METANAME_ANDROIDAUTO_TOUCHSCREEN_TYPE`
- `CINEMO_METANAME_ANDROIDAUTO_TOUCHSCREEN_X`
- `CINEMO_METANAME_ANDROIDAUTO_TOUCHSCREEN_Y`

A touchpad will be registered if all of the following metapool entries are provided:

- `CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_X`
- `CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_Y`
- `CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_X_PHYSICAL`
- `CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_Y_PHYSICAL`

Input feedback events will be registered for a display if an [ICinemoAndroidAutoInputCallbacks](#) interface is provided and the setup metapool contains valid `CINEMO_METANAME_ANDROIDAUTO_INPUT_FEEDBACK_EVENT` entries.

Media browser endpoint

The media browser endpoint will be registered if an [ICinemoAndroidAutoMediaBrowserCallbacks](#) interface is provided.

Media playback endpoint

The media playback endpoint will be registered if an [ICinemoAndroidAutoMediaPlaybackCallbacks](#) interface is provided.

Navigation endpoint

The navigation endpoint will be registered if an [ICinemoAndroidAutoNavigationCallbacks](#) interface is provided and the following metapool entry is present:

- CINEMO_METANAME_ANDROIDAUTO_NAVIGATION_MININTERVAL

Notification endpoint

The notification endpoint will be registered if an [ICinemoAndroidAutoNotificationCallbacks](#) interface is provided.

Phone endpoint

The phone endpoint will be registered if an [ICinemoAndroidAutoPhoneCallbacks](#) interface is provided.

Sensor endpoint

The sensor endpoint will be registered if at least one valid metapool entry for CINEMO_METANAME_ANDROIDAUTO_SENSORTYPE is found in the setup metapool.

Vendor extension endpoint

The vendor extension endpoint will be registered if an [ICinemoAndroidAutoVendorExtensionCallbacks](#) interface is provided and at least one vendor extension is defined in the setup metapool with the following entries:

- CINEMO_METANAME_ANDROIDAUTO_VEC_SERVICE_NAME
- CINEMO_METANAME_ANDROIDAUTO_VEC_PACKAGE_WHITE_LIST

Each CINEMO_METANAME_ANDROIDAUTO_VEC_SERVICE_NAME entry will create one endpoint.

Video sink endpoint

A video sink will be registered if all configured displays have a valid configuration in the setup metapool, containing at least the following entries:

- CINEMO_METANAME_ANDROIDAUTO_VIDEO_RESOLUTION
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_FPS

4.1.2 Compatibility with future Android Auto releases

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to these interfaces are likely.

4.1.3 Android Auto error codes reported by ICinemoPlayer

All errors related to an Android Auto session are reported via the [ICinemoAndroidCoreCallbacks::OnUnrecoverableError\(\)](#) callback. This includes errors from setting up a session as well as errors which occur during an active session. Each player, which is active at the time of the error, also generates an *CINEMO_EC_ERROR* event, but the associated error code there is always *CinemoErrorConnectionFailed* for all kinds of errors.

If a session has already started by the time an error occurs additional information about the reason can be obtained from the reason parameter of the [ICinemoAndroidCoreCallbacks::OnEndSession\(\)](#) callback.

For more information about specific error codes please see the respective sections about the individual callbacks.

4.1.4 Callback functions for Android Auto

Applications using the [ICinemoAndroidAuto](#) interface will be notified of Android Auto state changes and events by means of callback functions. All mandatory callback interfaces need to be provided in the [ICinemoAndroidAuto::SetCallbacks\(\)](#) method during the setup process. The additional callback interfaces, which are optional and may be omitted, can be registered with the [ICinemoAndroidAuto::SetExtraCallbacks\(\)](#) function. In order to pass Google certification tests it may be required to provide the additional interfaces as well.

Notes

An application should not make any assumptions about the order in which callback functions are called, or on which thread they are executed. Cinemo's Android Auto implementation runs multiple internal threads, all of which may deliver asynchronous events or notifications. An application must ensure its callback functions are capable of executing concurrently on multiple threads (for example, by implementing its own mutex or other queuing mechanism).

Executing any tasks or waiting for hardware events should not be performed in any of the callback functions. Doing so may lead to blocking of the basic worker threads of the Google receiver library. It is recommended to only extract the information carried by the callback function, return it afterwards and process the information in another thread (if possible).

4.2 Factory Functions for Android Auto

Functions to create built-in classes implementing the Android Auto interfaces.

These functions can be called globally and are not part of any Cinemo interface.

4.2.1 CinemoCreateAndroidAuto

Creates an instance of the built-in class that implements the [ICinemoAndroidAuto](#) interface.

Syntax

```
CinemoError CreateCinemoAndroidAuto(  
    [in/out] struct ICinemoAndroidAuto ** pp  
) ;
```

Parameters

- **pp**

Return an instance of the [ICinemoAndroidAuto](#) implementation

Return value

CinemoNoError on success.

4.3 ICinemoAndroidAuto

The [ICinemoAndroidAuto](#) interface documented here is part of a full implementation of Google's [Android Auto](#) projection protocol. While basic functionality for initialization and audio focus handling are exposed on this interface, the additional methods are available on respective dedicated interfaces.

Notes

The base interface, [ICinemoAndroidAuto](#), can be created with the `CreateCinemoAndroidAuto()` function. All other interfaces can be retrieved from the base [ICinemoAndroidAuto](#) object using the [ICinemoUnknown::QueryInterface\(\)](#) method.

ICinemoAndroidAuto Methods

METHOD	DESCRIPTION
Setup	Set the configuration metapool for the Android Auto session
SetCallbacks	Set mandatory callback interfaces for handling Android Auto sessions
SetExtraCallbacks	Set extra callback interfaces for Android Auto events
SetAudioFocus	Set the audio focus mode for audio transmitted from the Android device
SetNavigationFocus	Set the navigation focus mode for the Android device
ConfirmAudioPlaybackStart	Start audio after playback start has previously been delayed
SetAudioParamsMicrophone	Set audio device parameters for microphone stream
SetAudioParamsGuidance	Set audio device parameters for playback of guidance audio stream
SetAudioParamsMedia	Set audio device parameters for playback of media audio stream
SetCallAvailability	Sends call availability status to MD
DisconnectSession	Disconnect the currently active session
Open	Start either an Android Auto session with a USB device or a TCP server to accept incoming Android Auto wireless connection attempts
Close	Close the Android Auto object and terminate any active connection
GetDisplay	Get the display object associated with the specified video stream
GetMetapool	Get the Android Auto metapool containing some state information

Compatibility with future Android Auto releases

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to these interfaces are likely.

4.3.1 Setup

Set the configuration metapool for the Android Auto session.

Syntax

```
Setup(
    [in]     ICinemoMetapool * ppool
);
```

Parameters

- **ppool**

ICinemoMetapool containing Android Auto configuration information

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The information specified here will be used to set up the individual Android Auto interfaces like audio, video, input devices, the SSL certificates and much more. Cinemo currently recognizes the following metadata:

- CINEMO_METANAME_ANDROIDAUTO_MAKE
- CINEMO_METANAME_ANDROIDAUTO_MODEL
- CINEMO_METANAME_ANDROIDAUTO_YEAR
- CINEMO_METANAME_ANDROIDAUTO_ID
- CINEMO_METANAME_ANDROIDAUTO_HUMAKE
- CINEMO_METANAME_ANDROIDAUTO_HUMODEL
- CINEMO_METANAME_ANDROIDAUTO_HUSOFTWAREBUILD
- CINEMO_METANAME_ANDROIDAUTO_HUSOFTWAREVERSION
- CINEMO_METANAME_ANDROIDAUTO_ROOT_CERT
- CINEMO_METANAME_ANDROIDAUTO_CLIENT_CERT
- CINEMO_METANAME_ANDROIDAUTO_PRIVATE_KEY
- CINEMO_METANAME_ANDROIDAUTO_SESSIONCONFIGURATION
- CINEMO_METANAME_ANDROIDAUTO_DRIVERPOSITION
- CINEMO_METANAME_ANDROIDAUTO_CERT_VERIFICATION_TIME
- CINEMO_METANAME_ANDROIDAUTO_BLUETOOTH_ADDRESS
- CINEMO_METANAME_ANDROIDAUTO_BLUETOOTH_PAIRING¹
- CINEMO_METANAME_ANDROIDAUTO_TOUCHSCREEN_X
- CINEMO_METANAME_ANDROIDAUTO_TOUCHSCREEN_Y
- CINEMO_METANAME_ANDROIDAUTO_TOUCHSCREEN_TYPE
- CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_X
- CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_Y

- CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_X_PHYSICAL
- CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_Y_PHYSICAL
- CINEMO_METANAME_ANDROIDAUTO_KEYCODE¹
- CINEMO_METANAME_ANDROIDAUTO_NAVIGATION_MININTERVAL
- CINEMO_METANAME_ANDROIDAUTO_SENSORSOURCE_LOCATIONCHARACTERIZATION¹
- CINEMO_METANAME_ANDROIDAUTO_SENSORTYPE¹
- CINEMO_METANAME_TRANSPORT_USB_READ_SIZE
- CINEMO_METANAME_ANDROIDAUTO_VEC_SERVICE_NAME
- CINEMO_METANAME_ANDROIDAUTO_VEC_PACKAGE_WHITE_LIST
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_VIEWING_DISTANCE
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_RESOLUTION¹
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_FPS¹
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_DPI¹
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_DECODER_ADDITIONAL_DEPTH¹
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_AUTOSTART
- CINEMO_METANAME_AOAP_DETECTION_TIMEOUT
- CINEMO_METANAME_AOAP_SERIAL_NUMBER
- CINEMO_METANAME_ANDROIDAUTO_DISPLAYNAME
- CINEMO_METANAME_ANDROIDAUTO_PROBE_CONNECTION
- CINEMO_METANAME_ANDROIDAUTO_FUELTYPE¹
- CINEMO_METANAME_ANDROIDAUTO_EV_CONNECTOR_TYPE¹
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_REAL_DPI¹
- CINEMO_METANAME_ANDROIDAUTO_WIFI_BSSID
- CINEMO_METANAME_ANDROIDAUTO_TOUCHSCREEN_IS_SECONDARY
- CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_TAP_AS_SELECT
- CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_UI_ABSOLUTE
- CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_UI_NAVIGATION
- CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_UI_SENSITIVITY
- CINEMO_METANAME_ANDROIDAUTO_INPUT_FEEDBACK_EVENT¹
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_DISPLAY_TYPE²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_AUXILIARY_DISPLAY_TYPE
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_DISPLAY_ASPECT_X²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_DISPLAY_ASPECT_Y²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_MARGIN_LEFT²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_MARGIN_RIGHT²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_MARGIN_TOP²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_MARGIN_BOTTOM²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_CURRENT_CONTENT_INSET_LEFT²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_CURRENT_CONTENT_INSET_RIGHT²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_CURRENT_CONTENT_INSET_TOP²

- CINEMO_METANAME_ANDROIDAUTO_VIDEO_CURRENT_CONTENT_INSET_BOTTOM²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_CONTENT_INSETS_LEFT²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_CONTENT_INSETS_RIGHT²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_CONTENT_INSETS_TOP²
- CINEMO_METANAME_ANDROIDAUTO_VIDEO_CONTENT_INSETS_BOTTOM²

¹ multiple entries (with different indices) of this type may be present in the metapool

² multiple entries (with different title and entry indices) of this type may be present in the metapool.

Title index is responsible for the display index while the entry index is responsible for the configuration index. This method may only be called before calling the [ICinemoAndroidAuto::Open\(\)](#) method is called. If it is called while a session is in progress an error will be returned.

4.3.2 SetCallbacks

Set mandatory callback interfaces for handling Android Auto sessions.

Syntax

```
CinemoError SetCallbacks(  
    [in]     ICinemoAndroidAutoAudioCallbacks * paudio,  
    [in]     ICinemoAndroidAutoCoreCallbacks * pcore,  
    [in]     ICinemoAndroidAutoVideoCallbacks * pvideo  
) ;
```

Parameters

- **paudio**

pointer to audio callbacks, must not be nullptr

- **pcore**

pointer to core callbacks, must not be nullptr

- **pvideo**

pointer to video callbacks, must not be nullptr

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [SetCallbacks\(\)](#) method is used to specify application-defined callbacks to receive notifications from a running Android Auto session.

This method is synchronous and may only be called before starting playback of an Android Auto object. If it is called while a session is in progress an error will be returned.

4.3.3 SetExtraCallbacks

Set extra callback interfaces for Android Auto events.

Syntax

```
CinemoError SetExtraCallbacks (
    [in]     ICinemoAndroidAutoBluetoothCallbacks * pbluetooth,
    [in]     ICinemoAndroidAutoInputCallbacks * pinput,
    [in]     ICinemoAndroidAutoMediaBrowserCallbacks * pmediabrowser,
    [in]     ICinemoAndroidAutoMediaPlaybackCallbacks * pmediaplayback,
    [in]     ICinemoAndroidAutoNavigationCallbacks * pnavigation,
    [in]     ICinemoAndroidAutoNotificationCallbacks * pnotification,
    [in]     ICinemoAndroidAutoPhoneCallbacks * pphone,
    [in]     ICinemoAndroidAutoVendorExtensionCallbacks * pvendorextension,
    [in]     ICinemoAndroidAutoWifiCallbacks * pwifi
);
```

Parameters

- **pbluetooth**
pointer to bluetooth callbacks
- **pinput**
pointer to input callbacks
- **pmediabrowser**
pointer to media browser callbacks
- **pmediaplayback**
pointer to media playback callbacks
- **pnavigation**
pointer to navigation callbacks
- **pnotification**
pointer to notification callbacks
- **pphone**
pointer to phone callbacks
- **pvendorextension**
pointer to vendor extension callbacks
- **pwifi**
pointer to wifi callbacks

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The [SetExtraCallbacks\(\)](#) method is used to specify additional callback interfaces which are not mandatory for basic session operations but may be required for Google certification.

This method is synchronous and may only be called before starting playback of an Android Auto object. If it is called while a session is in progress an error will be returned.

4.3.4 SetAudioFocus

Set the audio focus mode for audio transmitted from the Android device.

Syntax

```
CinemoError SetAudioFocus (
    [in]      CINEMO_ANDROIDAUTO_AUDIOFOCUSSTATE state,
    [in]      uint32 unsolicited
);
```

Parameters

- **state**
Mode of the audio focus granted to the Android device
- **unsolicited**
Non-zero if this is an unsolicited message

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This function needs to be called after an [ICinemoAndroidAutoAudioCallbacks::OnAudioSinkFocusRequest\(\)](#) has been received or as an unsolicited message by the head unit.

In the Android Auto Head Unit Integration Guide from Google there are strict timing requirements regarding audio focus processing. It says that [ICinemoAndroidAutoAudioCallbacks::OnAudioSinkFocusRequest\(\)](#) shall be returned immediately without blocking. The following [SetAudioFocus\(\)](#) may be delayed by up to 500ms. If the audio setup will take longer, the audio focus should be granted latest 500ms after the focus request and the audio stream may be delayed by not sending ACKs for received audio frames. Cinemo provides two ways of implementing such a behavior.

The highly recommended method would be that all audio device related calls should be blocking until the device is ready. With this approach it is not required to implement special audio focus request handling and audio focus may always be granted immediately. Cinemo will be blocked creating and setting up the audio device. Only after these calls have returned audio frame ACKs will be sent to the Android device.

If it is absolutely not possible to block the calls for audio device creation, the Cinemo API may be used to delay the ACKs. For this method it is required to grant the audio focus before the 500 ms timeout. But when the [ICinemoAndroidAutoAudioCallbacks::OnAudioSinkStartPlayback\(\)](#) callback is called, *CinemoErrorPending* needs to be returned. This will block the creation of the audio device internally until [ConfirmAudioPlaybackStart\(\)](#) is called for the exact same stream as the playback start has been notified for. Please consult your Cinemo representative before using this function.

For more details on Android Auto audio focus negotiation see the Google documentation.

4.3.5 SetNavigationFocus

Set the navigation focus mode for the Android device.

Syntax

```
CinemoError SetNavigationFocus(  
    [in]      CINEMO_ANDROIDAUTO_NAVIGATIONFOCUSMODE mode  
) ;
```

Parameters

- **mode**

Mode of the navigation focus

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This function needs to be called after each [ICinemoAndroidAutoNavigationCallbacks::OnNavigationFocusRequest\(\)](#), even if there has been no change in navigation focus.

4.3.6 ConfirmAudioPlaybackStart

Start audio after playback start has previously been delayed.

Syntax

```
CinemoError ConfirmAudioPlaybackStart(  
    [in]      CINEMO_ANDROIDAUTO_AUDIOSTREAM stream  
) ;
```

Parameters

- **stream**

Stream which should be unblocked

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method is only required if the audio playback in the [ICinemoAndroidAutoAudioCallbacks::OnAudioSinkStartPlayback\(\)](#) callback has been delayed by returning *CinemoErrorPending*. Calling this method will internally unblock the audio handling and start sending ACKs for audio frames received for the channel specified by the stream argument. Details are described in the documentation of [SetAudioFocus\(\)](#).

Calling this method without previously returning *CinemoErrorPending* from the start callback will return *CinemoErrorState*.

4.3.7 SetAudioParamsMicrophone

Set audio device parameters for microphone stream.

Syntax

```
CinemoError SetAudioParamsMicrophone(  
    [in]      const CinemoAudioParams & params  
) ;
```

Parameters

- **params**

Audio device parameters

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method changes the parameters for platform-specific microphone input device.

This method may be called at any time except from within a callback from the registered callback interface. The provided parameters will become effective the next time the microphone is requested by the Android Device. For changing the default microphone device parameters, it is recommended to call this method before the Android Auto session is started.

4.3.8 SetAudioParamsGuidance

Set audio device parameters for playback of guidance audio stream.

Syntax

```
CinemoError SetAudioParamsGuidance(  
    [in]      const CinemoAudioParams & params  
) ;
```

Parameters

- **params**

Audio device parameters

Return value

Always *CinemoNoError*.

Remarks

This method changes the parameters for platform-specific audio output devices, and are functionally equivalent to [ICinemoPlayer::SetAudioParams\(\)](#). The only point of distinction is that [ICinemoAndroidAuto](#) exposes separate methods for configuring each audio stream independently.

This method may be called at any time except from within a callback from the registered callback interface. If they are called before an Android Auto session is started, the parameters will be saved and become effective when the corresponding playback device is next instantiated. If they are called during playback, the parameters will become effective immediately: as for [ICinemoPlayer](#), it is possible to dynamically change any attribute of the audio device, including whether it is enabled or disabled, during playback.

Note that audio output devices may be active simultaneously. This method is synchronous.

4.3.9 SetAudioParamsMedia

Set audio device parameters for playback of media audio stream.

Syntax

```
CinemoError SetAudioParamsMedia(  
    [in]      const CinemoAudioParams & params  
) ;
```

Parameters

- **params**

Audio device parameters

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method changes the parameters for platform-specific audio output devices, and are functionally equivalent to [ICinemoPlayer::SetAudioParams\(\)](#). The only difference is that [ICinemoAndroidAuto](#) exposes separate methods for configuring each audio stream independently.

This method may be called at any time except from within a callback from the registered callback interface. If they are called before an Android Auto session is started, the parameters will be stored and become active when the corresponding playback device is opened the next time. If they are called during playback, the parameters will become active immediately. As for the [ICinemoPlayer](#), it is possible to dynamically change any attribute of the audio device, including whether it is enabled or disabled, during playback.

Note that audio output devices may be active simultaneously. This method is synchronous.

4.3.10 SetCallAvailability

Sends call availability status to MD.

Syntax

```
CinemoError SetCallAvailability(  
    [in]      bool callAvailable  
) ;
```

Parameters

- **callAvailable**

Call availability status

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Sends call availability status to MD. The argument should be true if phone calls via HFP become available and false if phone calls via HFP become unavailable due to an ongoing call on another device/SIM.

4.3.11 DisconnectSession

Disconnect the currently active session.

Syntax

```
CinemoError DisconnectSession(  
    [in]      CINEMO_ANDROIDAUTO_SESSION_END_REASON reason  
) ;
```

Parameters

- **reason**
CINEMO_ANDROIDAUTO_SESSION_END_REASON enum

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will disconnect the currently active Android Auto session by sending a ByeByeRequest with the provided reason to the device. This may be used to disconnect wireless sessions without shutting down the Android Auto server.

For the reason parameter only enum values with a numeric value less than 100 are supported.

Currently those are:

- *CINEMO_ANDROIDAUTO_SESSION_END_USER_SELECTION*
- *CINEMO_ANDROIDAUTO_SESSION_END_NOT_SUPPORTED*
- *CINEMO_ANDROIDAUTO_SESSION_END_NOT_CURRENTLY_SUPPORTED*
- *CINEMO_ANDROIDAUTO_SESSION_END_PROBE_SUPPORTED*

In case an invalid reason is specified *CinemoErrorArguments* will be returned.

4.3.12 Open

Start either an Android Auto session with a USB device or a TCP server to accept incoming Android Auto wireless connection attempts.

Syntax

```
CinemoError Open(  
    [in]      const char * szurl  
) ;
```

Parameters

- **szurl**

URL of an Android device or a network address

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorArguments**

The provided URL has a wrong format or the setup metapool has not been provided yet.

- **CinemoErrorState**

The method has been called during an active session.

Remarks

To start Android Auto, the URL of the mobile device needs be provided as a parameter of this method. The URL must be a valid USB device path, starting with either “usb://” or “aoap://usb://”.

Some valid USB device paths might be the following:

- **Windows**

usb://000/001

- **Linux**

usb:///dev/bus/usb/003/005

- **QNX**

usb://0/000/001

This method can be called at any time after [ICinemoAndroidAuto::Setup\(\)](#) method is called. The playback of each of the display objects can be started also before [Open\(\)](#) is called.

4.3.13 Close

Close the Android Auto object and terminate any active connection.

Syntax

```
CinemoError Close();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This function should be used when the [ICinemoAndroidAuto](#) object is not needed anymore. Any active session will be terminated and the listening TCP socket will be closed.

4.3.14 GetDisplay

Get the display object associated with the specified video stream.

Syntax

```
CinemoError GetDisplay(  
    [out]     ICinemoAndroidAutoDisplay ** ppdisplay,  
    [in]      uint32 index  
) ;
```

Parameters

- **ppdisplay**

The address of a pointer that receives the requested display interface

- **index**

Index of the display to get

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorArguments**

The index provided is invalid

- **CinemoErrorResources**

The display object could not be allocated

- **CinemoErrorState**

The setup metapool has not yet been provided

Remarks

This function provides access to the [ICinemoAndroidAutoDisplay](#) objects, which are required for displaying the video streams by playing them with a player. The `index` parameter determines which display object will be obtained and must correspond to a metapool title index used for configuring this displays in the setup metapool.

This method can be called only after [ICinemoAndroidAuto::Setup\(\)](#) method has been called.

4.3.15 GetMetapool

Get the Android Auto metapool containing some state information.

Syntax

```
CinemoError GetMetapool(  
    [inout]  ICinemoMetapool * ppool  
) ;
```

Parameters

- **ppool**

The address of the metapool object which will be filled.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This function should be used to retrieve the Android Auto metapool. This should be done once a [ICinemoAndroidAutoCoreCallbacks::OnSessionEvent\(\)](#) callback has been received with the `event` parameter equal to `CINEMO_ANDROIDAUTO_EVENT_METAPOOL_UPDATED`.

Possible entries in the metapool are the following:

- `CINEMO_METANAME_VFS_BIND_PATH`
URI of the device which is currently connected.
- `CINEMO_METANAME_ANDROIDAUTO_WIFI_SERVER_ADDRESS`
URI of the WiFi server accepting incoming Android Auto connections.
- `CINEMO_METANAME_ANDROIDAUTO_VEC_SERVICE_NAME`
Name of the Vendor Extension Channel.
- `CINEMO_METANAME_ANDROIDAUTO_VEC_PATH`
Path of the Vendor Extension Channel.

4.4 ICinemoAndroidAutoBluetooth

The [ICinemoAndroidAutoBluetooth](#) interface is part of the Cinemo [Android Auto](#) implementation. While [ICinemoAndroidAuto](#) and [ICinemoAndroidAutoDisplay](#) implement audio and video focus related methods, the [ICinemoAndroidAutoBluetooth](#) interface is used to set up the Bluetooth connection between the head unit and the Android device.

This thread-safe interface needs to be queried from the base [ICinemoAndroidAuto](#) interface object:

```
CinemoAutoPtr<ICinemoAndroidAuto> androidauto;
CinemoAutoPtr<ICinemoAndroidAutoBluetooth> bluetooth;

CinemoCreateAndroidAuto(androidauto.pp());
androidauto.QueryInterface(bluetooth.pv(), ICinemoAndroidAutoBluetooth::iid);
```

ICinemoAndroidAutoBluetooth Methods

METHOD	DESCRIPTION
OnReadyForPairing	Signal the Android device to start Bluetooth pairing
RequestDelayedPairing	Delay the Bluetooth pairing process
SendAuthData	Send Bluetooth authentication data to the Android device

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

4.4.1 OnReadyForPairing

Signal the Android device to start Bluetooth pairing.

Syntax

```
CinemoError OnReadyForPairing (
    [in]      int alreadypaired
);
```

Parameters

- **alreadypaired**

Non-zero if the device is already paired with the head unit

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This function is part of the Bluetooth pairing process of the head unit and the Android device. Usually an [ICinemoAndroidAutoBluetoothCallbacks::OnBluetoothPairingRequest\(\)](#) is sent to the head unit, which must make itself ready to be paired with the phone. When the head unit is ready to start the Bluetooth pairing process, it needs to call [OnReadyForPairing\(\)](#).

4.4.2 RequestDelayedPairing

Delay the Bluetooth pairing process.

Syntax

```
CinemoError RequestDelayedPairing();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the head unit cannot be prepared for Bluetooth pairing within one second after receiving an [ICinemoAndroidAutoBluetoothCallbacks::OnBluetoothPairingRequest\(\)](#) callback, this function will delay the pairing process until [OnReadyForPairing\(\)](#) is called.

4.4.3 SendAuthData

Send Bluetooth authentication data to the Android device.

Syntax

```
CinemoError SendAuthData(  
    [in]      const char * szauthdata,  
    [in]      CINEMO_ANDROIDAUTO_BLUETOOTHPAIRING pairingMethod  
) ;
```

Parameters

- **sauthdata**

Authentication data

- **pairingMethod**

Pairing method reported by the HU's Bluetooth stack

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This methods sends authentication data, obtained from the Bluetooth stack, to the Android device. Using this data, the Android device will pair itself with the head unit. Pairing method provided in the second parameter may differ from the one reported by [ICinemoAndroidAutoBluetoothCallbacks::OnBluetoothAuthenticationResult\(\)](#).

4.5 ICinemoAndroidAutoDisplay

The [ICinemoAndroidAutoDisplay](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is for playback and focus control of the available video streams. All user input events like touch and key presses are also routed through this interface.

The display interface object, [ICinemoAndroidAutoDisplay](#), can be created with the [ICinemoAndroidAuto::GetDisplay\(\)](#) method.

ICinemoAndroidAutoDisplay Methods

METHOD	DESCRIPTION
ReportKey	Report a key press or release
ReportKeyLongPress	Report a long key press
ReportAbsolute	Send an input event from a device with absolute values
ReportRelative	Send an input event from a device with relative values
ReportTouch	
ReportTouchPad	Report an input event from the touch pad
SendKeyEvent	Report a key press or release
SendTouchscreenEvent	Report an input event from the touch screen
UpdateTouchPadSensitivity	Update the touchpad sensitivity setting
SetVideoFocus	Set video focus mode
SetMargins	Set margins for a video display
SetContentInsets	Set content insets for a video display
SetStableContentInsets	Set stable content insets for a video display
GetMargins	Get margins for a video display
GetContentInsets	Get content insets for a video display
GetStableContentInsets	Get stable content insets for a video display

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

4.5.1 ReportKey

Report a key press or release.

Syntax

```
CinemoError ReportKey(  
    [in]      uint64 timestamp,  
    [in]      uint32 key,  
    [in]      int is_down,  
    [in]      uint32 meta  
) ;
```

Parameters

- **timestamp**

Timestamp of the event in nanoseconds

- **key**

Keycode of the key

- **is_down**

Non-zero if it is a key press, zero for key release

- **meta**

Can be used to represent shift, alt etc. Currently unused.

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorArguments**

The specified keycode is invalid.

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

This method should be used to report a key press or release. It is the preferred interface for signaling key events.

4.5.2 ReportKeyLongPress

Report a long key press.

Syntax

```
CinemoError ReportKeyLongPress(  
    [in]      uint64 timestamp,  
    [in]      uint32 key,  
    [in]      uint32 meta  
) ;
```

Parameters

- **timestamp**

Timestamp of the event in nanoseconds

- **key**

Keycode of the key

- **meta**

Can be used to represent shift, alt etc. Currently unused.

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorArguments**

The specified keycode is invalid.

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

This method should be used to report a long key press. What this does is simulate a sequence of key down, hold, key up on the phone. The only case when you should do this is if your system prevents you from being able to detect the initial key down. Do not report a key up after this.

4.5.3 ReportAbsolute

Send an input event from a device with absolute values.

Syntax

```
CinemoError ReportAbsolute(  
    [in]      uint64 timestamp,  
    [in]      uint32 key,  
    [in]      int value  
) ;
```

Parameters

- **timestamp**

Timestamp of the event in nanoseconds

- **key**

Keycode of the key

- **value**

The absolute value associated with that device

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorArguments**

The specified keycode is invalid.

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

When using an input device which sends absolute values, this function can be used to report them to the Android device. While these devices aren't keys in the common sense, they use a keycode as a device identifier, which needs to be registered using *CINEMO_METANAME_ANDROIDAUTO_KEYCODE* in the setup metapool.

4.5.4 ReportRelative

Send an input event from a device with relative values.

Syntax

```
CinemoError ReportRelative(
    [in]      uint64 timestamp,
    [in]      uint32 key,
    [in]      int delta
);
```

Parameters

- **timestamp**

Timestamp of the event in nanoseconds

- **key**

Keycode of the key

- **delta**

The value associated with this event

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorArguments**

The specified keycode is invalid.

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

When using an input device which sends relative values, this function can be used to report them to the Android device. While these devices aren't keys in the common sense, they use a keycode as a device identifier, which need to be registered using *CINEMO_METANAME_ANDROIDAUTO_KEYCODE* in the metapool. You can indicate a magnitude and direction using the value and its sign. For instance, if a jog dial were turned two steps to the left, delta would be -2. If it were turned 5 steps to the right, delta would be +5.

4.5.5 ReportTouch

Report an input event from the touch screen.

Syntax

```
CinemoError ReportTouch(
    [in]      uint64 timestamp,
    [in]      size_t numPointers,
    [in]      const uint32 * pointerIDs,
    [in]      const uint32 * x,
    [in]      const uint32 * y,
    [in]      int action,
    [in]      int actionIndex
);
```

Parameters

- **timestamp**

Timestamp of the event in nanoseconds

- **numPointers**

The number of pointers being described

- **pointerIDs**

The pointer IDs of the pointers

- **x**

The x values of the pointers

- **y**

The y values of the pointers

- **action**

The action associated with this event

- **actionIndex**

The index of the pointer to which the action belongs

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

Use this method to report a touch event on the touch screen that is associated with that display. Coordinates reported must be translated and/or scaled as appropriate such that they are within the coordinate space of the display and so if the display is running a scaled resolution, the x and y values should be scaled appropriately too.

HUMAX CONFIDENTIAL AUTOMOTIVE

4.5.6 ReportTouchPad

Report an input event from the touch pad.

Syntax

```
CinemoError ReportTouchPad(
    [in]      uint64 timestamp,
    [in]      size_t numPointers,
    [in]      const uint32 * pointerIDs,
    [in]      const uint32 * x,
    [in]      const uint32 * y,
    [in]      int action,
    [in]      int actionIndex
);
```

Parameters

- **timestamp**

Timestamp of the event in nanoseconds

- **numPointers**

The number of pointers being described

- **pointerIDs**

The pointer IDs of the pointers

- **x**

The x values of the pointers

- **y**

The y values of the pointers

- **action**

The action associated with this event

- **actionIndex**

The index of the pointer to which the action belongs

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

Use this method to report a touch event on the touch pad. The coordinates must be absolute and within the coordinate space of the touch pad itself.

4.5.7 SendKeyEvent

Report a key press or release.

Syntax

```
CinemoError SendKeyEvent(  
    [in]      uint32 key,  
    [in]      int is_down,  
    [in]      uint32 meta  
) ;
```

Parameters

- **key**

Keycode of the key

- **is_down**

Non-zero if it is a key press, zero for key release

- **meta**

Can be used to represent shift, alt etc. Currently unused.

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorArguments**

The specified keycode is invalid.

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

This method is identical to the `ReportKey()`, except for the missing `timestamp` argument. Cinemo will create an appropriate timestamp internally.

4.5.8 SendTouchscreenEvent

Report an input event from the touch screen.

Syntax

```
CinemoError SendTouchscreenEvent(  
    [in]      const CinemoAndroidAutoTouchPoint * ppoints,  
    [in]      size_t npoints,  
    [in]      CINEMO_ANDROIDAUTO_POINTERACTION action,  
    [in]      uint32 actionIndex  
) ;
```

Parameters

- **ppoints**

Pointer to CinemoAndroidAutoTouchPoints

- **npoints**

Number of items in ppoints

- **action**

The action that took place in this event

- **actionIndex**

The index of the pointer to which the action belongs

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

This method is identical to the ReportTouchScreen(), except for different arrangement of the arguments. Cinemo will create an appropriate timestamp internally and use the CinemoAndroidAutoTouchPoint structure.

4.5.9 UpdateTouchPadSensitivity

Update the touchpad sensitivity setting.

Syntax

```
CinemoError UpdateTouchPadSensitivity(  
    [in]      uint32 sensitivity  
) ;
```

Parameters

- **sensitivity**

Sensitivity level of the touchpad

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

The sensitivity of the touchpad can be any number from 1 to 10 (inclusive). Larger numbers are more sensitive, requiring smaller inputs to trigger movement. This only applies when CINEMO_METANAME_ANDROIDAUTO_TOUCHPAD_UI_NAVIGATION is set to 1 and does not affect usages like handwriting input.

4.5.10 SetVideoFocus

Set video focus mode.

Syntax

```
CinemoError SetVideoFocus(  
    [in]      CINEMO_ANDROIDAUTO_VIDEOFOCUSMODE mode,  
    [in]      int unsolicited  
) ;
```

Parameters

- **mode**

Mode of the video focus

- **unsolicited**

Non-zero if this is an unsolicited message

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

This function needs to be called if the video focus changes in an unsolicited manner or an

[ICinemoAndroidAutoVideoCallbacks::OnVideoFocusRequest\(\)](#) comes through.

4.5.11 SetMargins

Set margins for a video display.

Syntax

```
CinemoError SetMargins(  
    [in]      const CinemoAndroidAutoInsets & margins  
) ;
```

Parameters

- **margins**

Reference to a structure containing the margins

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

The method allows to set the top, bottom, left and right insets using a *CinemoAndroidAutoInsets* structure.

Once the insets become active for the current video stream the [ICinemoAndroidAutoVideoCallbacks::OnVideoUiConfigChange\(\)](#) callback will be called. The parameters will contain the new insets and the index of the display, for which this change occurred. For more information please refer to the respective section of the documentation.

4.5.12 SetContentInsets

Set content insets for a video display.

Syntax

```
CinemoError SetContentInsets(  
    [in]      const CinemoAndroidAutoInsets & insets  
) ;
```

Parameters

- **insets**

Reference to a structure containing the content insets

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

The method allows to set the top, bottom, left and right content insets using the *CinemoAndroidAutoInsets* structure.

Once the insets become active for the current video stream the [ICinemoAndroidAutoVideoCallbacks::OnVideoUiConfigChange\(\)](#) callback will be called. The parameters will contain the new insets and the index of the display, for which this change occurred. For more information please refer to the respective section of the documentation.

4.5.13 SetStableContentInsetss

Set stable content insets for a video display.

Syntax

```
CinemoError SetStableContentInsetss(  
    [in]      const CinemoAndroidAutoInsetss & stableinsets  
) ;
```

Parameters

- **stableinsets**

Reference to a structure containing the stable content insets

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

The method allows to set the top, bottom, left and right stable content insets using the *CinemoAndroidAutoInsetss* structure.

Once the insets become active for the current video stream the [ICinemoAndroidAutoVideoCallbacks::OnVideoUiConfigChange\(\)](#) callback will be called. The parameters will contain the new insets and the index of the display, for which this change occurred. For more information please refer to the respective section of the documentation.

4.5.14 GetMargins

Get margins for a video display.

Syntax

```
CinemoError GetMargins(  
    [out]     CinemoAndroidAutoInsets & margins  
) ;
```

Parameters

- **margins**

Reference to a structure to be filled with the current margins

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

The method allows to get the top, bottom, left and right margins using the *CinemoAndroidAutoInsets* structure.

4.5.15 GetContentInsets

Get content insets for a video display.

Syntax

```
CinemoError GetContentInsets(  
    [out]     CinemoAndroidAutoInsets & insets  
) ;
```

Parameters

- **insets**

Reference to a structure to be filled with the current insets

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

The method allows to get the top, bottom, left and right insets using the *CinemoAndroidAutoInsets* structure.

4.5.16 GetStableContentInsets

Get stable content insets for a video display.

Syntax

```
CinemoError GetStableContentInsets(  
    [out]     CinemoAndroidAutoInsets & stableinsets  
) ;
```

Parameters

- **stableinsets**

Reference to a structure to be filled with the current stable insets

Return value

CinemoNoError on success. Otherwise it will return one of the following error codes:

- **CinemoErrorNotSupported**

The display does not support input actions. This may be the case for non-interactive auxiliary displays.

- **CinemoErrorState**

Returned when called without an active Android Auto session.

Remarks

The method allows to get the top, bottom, left and right stable insets using the *CinemoAndroidAutoInsets* structure.

4.6 ICinemoAndroidAutoMediaBrowser

The [ICinemoAndroidAutoMediaBrowser](#) interface is part of the Cinemo [Android Auto](#) implementation. While [ICinemoAndroidAuto](#) and [ICinemoAndroidAutoDisplay](#) implement audio and video focus related methods, the [ICinemoAndroidAutoMediaBrowser](#) interface can be used to browse media files available on the Android device.

This thread-safe interface needs to be queried from the base [ICinemoAndroidAuto](#) interface object:

```
CinemoAutoPtr<ICinemoAndroidAuto> androidauto;
CinemoAutoPtr<ICinemoAndroidAutoMediaBrowser> mediabrowser;

CinemoCreateAndroidAuto(androidauto.pp());
androidauto.QueryInterface(mediabrowser.pv(), ICinemoAndroidAutoMediaBrowser::iid);
```

ICinemoAndroidAutoMediaBrowser Methods

METHOD	DESCRIPTION
GetNode	Fetch a given media node
ReportAction	Report user input

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

4.6.1 GetNode

Fetch a given media node.

Syntax

```
CinemoError GetNode(  
    [in]      const char * szpath,  
    [in]      int start,  
    [in]      int get_album_art  
) ;
```

Parameters

- **szpath**

Full path of the node

- **start**

If this node is a List or Source node, start listing leaves at this index

- **get_album_art**

If non-zero, return album art if available

Return value

If the method succeeds, it returns CinemoNoError. Otherwise, it returns a CinemoError code.

Remarks

This method sends a media node request to the Android device, which responds with the corresponding media node callback.

4.6.2 ReportAction

Report user input.

Syntax

```
CinemoError ReportAction(  
    [in]      const char * szpath,  
    [in]      CINEMO_ANDROIDAUTO_INPUT action  
) ;
```

Parameters

- **szpath**

Full path of the node

- **action**

User input

Return value

If the method succeeds, it returns CinemoNoError. Otherwise, it returns a CinemoError code.

Remarks

If the specified action is invalid, CinemoErrorArguments is returned.

4.7 ICinemoAndroidAutoMediaPlayback

The [ICinemoAndroidAutoMediaBrowser](#) interface is part of the Cinemo [Android Auto](#) implementation. While [ICinemoAndroidAuto](#) and [ICinemoAndroidAutoDisplay](#) implement audio and video focus related methods, the [ICinemoAndroidAutoMediaPlayback](#) interface can be used to access media playback status information.

This thread-safe interface needs to be queried from the base [ICinemoAndroidAuto](#) interface object:

```
CinemoAutoPtr<ICinemoAndroidAuto> androidauto;
CinemoAutoPtr<ICinemoAndroidAutoMediaPlayback> mediaplayback;

CinemoCreateAndroidAuto(androidauto.pp());
androidauto.QueryInterface(mediaplayback.pv(), ICinemoAndroidAutoMediaPlayback::iid);
```

ICinemoAndroidAutoMediaPlayback Methods

METHOD	DESCRIPTION
ReportAction	Report user input

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

4.7.1 ReportAction

Report user input.

Syntax

```
CinemoError ReportAction(  
    [in]      CINEMO_ANDROIDAUTO_INPUT action  
) ;
```

Parameters

- **action**

User input

Return value

If the method succeeds, it returns CinemoNoError. Otherwise, it returns a CinemoError code.

Remarks

If the specified action is invalid, CinemoErrorArguments is returned.

4.8 ICinemoAndroidAutoNavigation

The [ICinemoAndroidAutoNavigation](#) interface is part of the Cinemo [Android Auto](#) implementation. While the base interface [ICinemoAndroidAuto](#) implements audio and video focus related methods, the [ICinemoAndroidAutoNavigation](#) interface can be used to access navigation data for the instrument cluster.

This thread-safe interface needs to be queried from the base [ICinemoAndroidAuto](#) interface object:

```
CinemoAutoPtr<ICinemoAndroidAuto> androidauto;
CinemoAutoPtr<ICinemoAndroidAutoNavigation> navigation;

CinemoCreateAndroidAuto(androidauto.pp());
androidauto.QueryInterface(navigation.pv(), ICinemoAndroidAutoNavigation::iid);
```

ICinemoAndroidAutoNavigation Methods

METHOD	DESCRIPTION
Start	Request the Android device to start sending navigation data
Stop	Request the Android device to stop sending navigation data

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

4.8.1 Start

Request the Android device to start sending navigation data.

Syntax

```
CinemoError Start();
```

Return value

This method returns *CinemoNoError*.

4.8.2 Stop

Request the Android device to stop sending navigation data.

Syntax

```
CinemoError Stop();
```

Return value

This method returns *CinemoNoError*.

4.9 ICinemoAndroidAutoNotification

The [ICinemoAndroidAutoNotification](#) interface is part of the Cinemo [Android Auto](#) implementation. While the base interface [ICinemoAndroidAuto](#) implements audio and video focus related methods, the [ICinemoAndroidAutoNotification](#) interface can be used to exchange freeform text notifications between the Android device and the head unit.

This thread-safe interface needs to be queried from the base [ICinemoAndroidAuto](#) interface object:

```
CinemoAutoPtr<ICinemoAndroidAuto> androidauto;  
CinemoAutoPtr<ICinemoAndroidAutoNotification> notification;  
  
CinemoCreateAndroidAuto(androidauto.pp());  
androidauto.QueryInterface(notification.pv(), ICinemoAndroidAutoNotification::iid);
```

ICinemoAndroidAutoNotification Methods

METHOD	DESCRIPTION
Ack	Acknowledge a notification sent by the Android device
Send	Send a notification to the Android device

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

4.9.1 Ack

Acknowledge a notification sent by the Android device.

Syntax

```
CinemoError Ack(  
    [in]      const char * szid  
) ;
```

Parameters

- **szid**

The identifier for the notification.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise it will return one of the following error codes:

- **CinemoErrorFailed:**

The communication channel is not yet subscribed.

Remarks

Calling this method informs the other side that the notification has been communicated to the user (displayed, read aloud etc.).

4.9.2 Send

Send a notification to the Android device.

Syntax

```
CinemoError Send(
    [in]      const char * sztext,
    [in]      int hasId,
    [in]      const char * szid,
    [in]      int hasIcon,
    [in]      const uchar * icon,
    [in]      size_t len
);
```

Parameters

- **sztext**

The text of the notification to display.

- **hasId**

If true, the id parameter contains valid data.

- **szid**

An identifier for this notification.

- **hasIcon**

If true, the icon argument contains valid data.

- **icon**

A pointer to the icon data (png format).

- **len**

The size of the icon in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise it will return one of the following error codes:

- **CinemoErrorFailed:**

The communication channel is not yet subscribed.

Remarks

An acknowledgment for receiving this message is only sent from the Android device if the ID for this notification is set. Otherwise no acknowledgment will be sent.

The size of the icon can be 96x96 pixels at maximum.

4.10 ICinemoAndroidAutoPhone

The [ICinemoAndroidAutoPhone](#) interface is part of the Cinemo [Android Auto](#) implementation. While the base interface [ICinemoAndroidAuto](#) implements audio and video focus related methods, the [ICinemoAndroidAutoPhone](#) interface can be used to access the phone status.

This thread-safe interface needs to be queried from the base [ICinemoAndroidAuto](#) interface object:

```
CinemoAutoPtr<ICinemoAndroidAuto> androidauto;
CinemoAutoPtr<ICinemoAndroidAutoPhone> phone;

CinemoCreateAndroidAuto(androidauto.pp());
androidauto.QueryInterface(phone.pv(), ICinemoAndroidAutoPhone::iid);
```

ICinemoAndroidAutoPhone Methods

METHOD	DESCRIPTION
ReportAction	Report user input

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

4.10.1 ReportAction

Report user input.

Syntax

```
CinemoError ReportAction(  
    [in]      const char * szcallernumber,  
    [in]      const char * szcallerid,  
    [in]      CINEMO_ANDROIDAUTO_INPUT action  
) ;
```

Parameters

- **szcallernumber**

The number of the call to act upon. Can be empty.

- **szcallerid**

The caller id to act upon. Can be empty.

- **action**

User action.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise it will return one of the following error codes:

- **CinemoErrorArguments:**

The specified keycode is not bound.

Remarks

The action field may take one of the following values:

- CINEMO_ANDROIDAUTO_INPUT_UP
- CINEMO_ANDROIDAUTO_INPUT_DOWN
- CINEMO_ANDROIDAUTO_INPUT_LEFT
- CINEMO_ANDROIDAUTO_INPUT_RIGHT
- CINEMO_ANDROIDAUTO_INPUT_ENTER
- CINEMO_ANDROIDAUTO_INPUT_BACK
- CINEMO_ANDROIDAUTO_INPUT_CALL

4.11 ICinemoAndroidAutoSensor

The [ICinemoAndroidAutoSensor](#) interface is part of the Cinemo [Android Auto](#) implementation. While the base interface [ICinemoAndroidAuto](#) implements audio and video focus related methods, the [ICinemoAndroidAutoSensor](#) interface can be used to send sensor data to the Android device.

This thread-safe interface needs to be queried from the base [ICinemoAndroidAuto](#) interface object:

```
CinemoAutoPtr<ICinemoAndroidAuto> androidauto;
CinemoAutoPtr<ICinemoAndroidAutoSensor> sensor;

CinemoCreateAndroidAuto(androidauto.pp());
androidauto.QueryInterface(sensor.pv(), ICinemoAndroidAutoSensor::iid);
```

ICinemoAndroidAutoSensor Methods

METHOD	DESCRIPTION
<i>ReportAccelerometer</i>	Report data from an accelerometer
<i>ReportCompass</i>	Report the current value of an on-board magnetic compass
<i>ReportCompass3D</i>	Report the current car orientation.
<i>ReportDoor</i>	Report door status
<i>ReportDrivingStatus</i>	Report driving status
<i>ReportEnvironment</i>	Report external environment data
<i>ReportFuel</i>	Report amount of fuel remaining
<i>ReportGear</i>	Report the current gear
<i>ReportGpsSatellite</i>	Report GPS satellite data
<i>ReportGyroscope</i>	Report data from a gyroscope
<i>ReportHvac</i>	Report HVAC data
<i>ReportLight</i>	Report status of lights
<i>ReportLocation</i>	Report a GPS location
<i>ReportNightMode</i>	Report value of the day-night sensor
<i>ReportOdometer</i>	Report the current odometer value
<i>ReportParkingBrake</i>	Report state of the parking brake
<i>ReportPassenger</i>	Report passenger presence
<i>ReportRPM</i>	Report the current RPM of the engine
<i>ReportSpeed</i>	Report the speed of the car
<i>ReportTirePressure</i>	Report tire pressure
<i>ReportTollCardData</i>	Report toll card data
<i>ReportError</i>	Report sensor error

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

HUMAX CONFIDENTIAL AUTOMOTIVE

4.11.1 ReportAccelerometer

Report data from an accelerometer.

Syntax

```
CinemoError ReportAccelerometer(  
    [in]      int hasaccelerationX,  
    [in]      int xaccelerationE3,  
    [in]      int hasaccelerationY,  
    [in]      int yaccelerationE3,  
    [in]      int hasaccelerationZ,  
    [in]      int zaccelerationE3  
) ;
```

Parameters

- **hasaccelerationX**

Acceleration along X-axis is valid.

- **xaccelerationE3**

Acceleration from left door to right door.

- **hasaccelerationY**

Acceleration along Y-axis is valid.

- **yaccelerationE3**

Acceleration from back to nose.

- **hasaccelerationZ**

Acceleration along Z-axis is valid.

- **zaccelerationE3**

Acceleration from floor to ceiling.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Use this call to report data from an accelerometer (include gravity). Units are m/s² multiplied by 1e3.

Acceleration data is mandatory if available and should be updated with a frequency of 10 Hz.

4.11.2 ReportCompass

Report the current value of an on-board magnetic compass.

Syntax

```
CinemoError ReportCompass(  
    [in]      int bearingE6  
) ;
```

Parameters

- **bearingE6**

The compass bearing [0, 360) multiplied by 1e6.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

The value reported here might be used in dead reckoning the position of the vehicle in the event of a GPS signal loss.

1-axis compass heading. Compass heading updates differ from the course in a location update:

- The heading can be updated independently of location information, enabling compass updates when no accurate location is available (such as in tunnels).
- The heading always indicates the direction of the front of the vehicle, whereas location course indicates the direction of movement (velocity vector). These are usually the same when traveling forwards, but are 180 degrees apart when reversing.

The compass heading is the orientation of the vehicle as determined by magnetic sensors; location course is the direction of movement (velocity vector) as determined by the location engine. If a head unit does not have access to a compass heading that is independent of the location course then compass updates should not be sent. The compass heading should be magnetic north (rather than true north). The recommended resolution of compass heading data is 1 degree, and the minimum resolution is 45 degrees.

Compass data is mandatory if available and should be updated with a frequency of 10 Hz.

4.11.3 ReportCompass3D

Report the current car orientation.

Syntax

```
CinemoError ReportCompass3D(
    [in]      int bearingE6,
    [in]      int haspitch,
    [in]      int pitchE6,
    [in]      int hasroll,
    [in]      int rollE6
);
```

Parameters

- **bearingE6**

The compass bearing [0, 360) multiplied by 1e6.

- **haspitch**

True if pitch parameter provided is valid.

- **pitchE6**

Car's pitch [-90, 90] multiplied by 1e6. Nose down is positive.

- **hasroll**

True if roll parameter provided is valid.

- **rollE6**

Car's roll (-180, 180] multiplied by 1e6. Right door down is positive.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

The value reported here might be used in dead reckoning the position of the vehicle in the event of a GPS signal loss.

3-axis compass heading if pitch and roll are available. Compass heading updates differ from the course in a location update:

- The heading can be updated independently of location information, enabling compass updates when no accurate location is available (such as in tunnels).
- The heading always indicates the direction of the front of the vehicle, whereas location course indicates the direction of movement (velocity vector). These are usually the same when traveling forwards, but are 180 degrees apart when reversing.

The compass heading is the orientation of the vehicle as determined by magnetic sensors; location course is the direction of movement (velocity vector) as determined by the location engine. If a head unit does not have

access to a compass heading that is independent of the location course then compass updates should not be sent. The compass heading should be magnetic north (rather than true north). The recommended resolution of compass heading data is 1 degree, and the minimum resolution is 45 degrees.

Compass data is mandatory if available and should be updated with a frequency of 10 Hz.

4.11.4 ReportDoor

Report door status.

Syntax

```
CinemoError ReportDoor(  
    [in]      int hashoodopen,  
    [in]      int hoodopen,  
    [in]      int hastrunkopen,  
    [in]      int trunkopen,  
    [in]      const int * dooropen,  
    [in]      size_t dooropenentries  
) ;
```

Parameters

- **hashoodopen**

Is the hoodopen parameter valid.

- **hoodopen**

True if the hood is open, false otherwise.

- **hastrunkopen**

Is the trunkopen parameter valid.

- **trunkopen**

True if the trunk is open, false otherwise.

- **dooropen**

Array of door states, true if the door is open, false otherwise.

- **dooropenentries**

Number of elements in the door open array.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Door status is optional and should be updated on change.

4.11.5 ReportDrivingStatus

Report driving status.

Syntax

```
CinemoError ReportDrivingStatus(
    [in]      int drivingstatus
);
```

Parameters

- **drivingstatus**

A bitmask of restrictions currently in effect.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

The following flags can be set for the drivingstatus:

- CINEMO_ANDROIDAUTO_DRIVESTATUS_UNRESTRICTED
- CINEMO_ANDROIDAUTO_DRIVESTATUS_NO_VIDEO
- CINEMO_ANDROIDAUTO_DRIVESTATUS_NO_KEYBOARD_INPUT
- CINEMO_ANDROIDAUTO_DRIVESTATUS_NO_VOICE_INPUT
- CINEMO_ANDROIDAUTO_DRIVESTATUS_NO_CONFIG
- CINEMO_ANDROIDAUTO_DRIVESTATUS_LIMIT_MESSAGE_LEN

Remarks

Indicates the level of feature/functionality lockout as determined by the vehicle. Defined as a bitmask:

CINEMO_ANDROIDAUTO_DRIVESTATUS_NO_VIDEO = no video playback allowed (video playback is defined as media such as movies, YouTube, games, etc. – not UI).

CINEMO_ANDROIDAUTO_DRIVESTATUS_NO_KEYBOARD_INPUT = no text input allowed (on-screen keyboard, rotary controller speller, touchpad text entry)

CINEMO_ANDROIDAUTO_DRIVESTATUS_NO_VOICE_INPUT = no voice input allowed

CINEMO_ANDROIDAUTO_DRIVESTATUS_NO_CONFIG = no setup/configuration allowed

CINEMO_ANDROIDAUTO_DRIVESTATUS_LIMIT_MESSAGE_LEN = limit displayed message length

Driving status data is mandatory and should be updated on change.

4.11.6 ReportEnvironment

Report external environment data.

Syntax

```
CinemoError ReportEnvironment (
    [in]      int hastemperature,
    [in]      int temperatureE3,
    [in]      int haspressure,
    [in]      int pressureE3,
    [in]      int hasrain,
    [in]      int rain
);
```

Parameters

- **hastemperature**

Is the temperatureE3 value valid.

- **temperatureE3**

The temperature in Celsius multiplied by 1e3.

- **haspressure**

Is the pressure parameter value valid.

- **pressureE3**

Pressure in kPa multiplied by 1e3.

- **hasrain**

Is the rain parameter value valid.

- **rain**

Rain detection level. 0 means no rain.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Environmental data is optional and should be updated on change.

4.11.7 ReportFuel

Report amount of fuel remaining.

Syntax

```
CinemoError ReportFuel(  
    [in]      int hasfuelremainingpercent,  
    [in]      int fuelremainingpercent,  
    [in]      int hasrangekm,  
    [in]      int rangekm,  
    [in]      int haslowfuelwarning,  
    [in]      int lowfuelwarning  
) ;
```

Parameters

- **hasfuelremainingpercent**

True if the fuelremainingpercent parameter should be considered, false otherwise.

- **fuelremainingpercent**

Fuel remaining in whole number percent values.

- **hasrangekm**

True if the rangekm parameter should be considered, false otherwise.

- **rangekm**

The estimated remaining range for the current amount of fuel.

- **haslowfuelwarning**

True if the lowfuelwarning parameter is valid.

- **lowfuelwarning**

True if the low fuel warning is on, false otherwise.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

This data may be used as supplementary input to navigation applications.

Fuel data is optional and should be updated on change.

4.11.8 ReportGear

Report the current gear.

Syntax

```
CinemoError ReportGear(  
    [in]      CINEMO_ANDROIDAUTO_GEAR gear  
) ;
```

Parameters

- **gear**

Current gear.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

The value of this sensor may be used to determine which UI elements can be interacted with and which ones get locked out. Additionally, these values might be used informationally.

The `gear` field may take one of the following values:

- CINEMO_ANDROIDAUTO_GEAR_NEUTRAL
- CINEMO_ANDROIDAUTO_GEAR_1
- CINEMO_ANDROIDAUTO_GEAR_2
- CINEMO_ANDROIDAUTO_GEAR_3
- CINEMO_ANDROIDAUTO_GEAR_4
- CINEMO_ANDROIDAUTO_GEAR_5
- CINEMO_ANDROIDAUTO_GEAR_6
- CINEMO_ANDROIDAUTO_GEAR_7
- CINEMO_ANDROIDAUTO_GEAR_8
- CINEMO_ANDROIDAUTO_GEAR_9
- CINEMO_ANDROIDAUTO_GEAR_10
- CINEMO_ANDROIDAUTO_GEAR_DRIVE
- CINEMO_ANDROIDAUTO_GEAR_PARK
- CINEMO_ANDROIDAUTO_GEAR_REVERSE

Current gear data is mandatory and should be updated on change.

4.11.9 ReportGpsSatellite

Report GPS satellite data.

Syntax

```
CinemoError ReportGpsSatellite(
    [in]      int numberinuse,
    [in]      int hasnumberinview,
    [in]      int numberinview,
    [in]      const int * prns,
    [in]      const int * snrs,
    [in]      const int * usedinfix,
    [in]      const int * azimuthsE3,
    [in]      const int * elevationsE3
);
```

Parameters

- **numberinuse**

Number of satellites used in GPS fix.

- **hasnumberinview**

Whether numberinview is valid.

- **numberinview**

Number of satellites visible to the GPS receiver.

- **prns**

Array of PRNs of satellites in view or NULL if per-satellite info unavailable.

- **snrs**

Array of SNRs of satellites in view in dB multiplied by 1e3 or NULL if per-satellite info unavailable.

- **usedinfix**

Array of flags whether this satellite was used in GPS fix or NULL if per-satellite info unavailable.

- **azimuthsE3**

Array of azimuths of satellites in degrees clockwise from north multiplied by 1e3 or NULL if per-satellite info unavailable or position data for satellites is absent.

- **elevationsE3**

Array of elevations of satellites in degrees from horizon to zenith multiplied by 1e3 or NULL if per-satellite info unavailable or position data for satellites is absent.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

At least number in-use must be available. If per-satellite info is available then arrays must contain numberinview elements.

The update frequency for this data is unknown.

HUMAX CONFIDENTIAL

4.11.10 ReportGyroscope

Report data from a gyroscope.

Syntax

```
CinemoError ReportGyroscope(  
    [in]      int hasrotationSpeedX,  
    [in]      int xrotationSpeedE3,  
    [in]      int hasrotationSpeedY,  
    [in]      int yrotationSpeedE3,  
    [in]      int hasrotationSpeedZ,  
    [in]      int zrotationSpeedE3  
) ;
```

Parameters

- **hasrotationSpeedX**

Rotation speed around X-axis is valid.

- **xrotationSpeedE3**

Rotation speed around axis from left door to right.

- **hasrotationSpeedY**

Rotation speed around Y-axis is valid.

- **yrotationSpeedE3**

Rotation speed around axis from back to nose.

- **hasrotationSpeedZ**

Rotation speed around Z-axis is valid.

- **zrotationSpeedE3**

Rotation speed around axis from floor to ceiling.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Units are rad/s multiplied by 1e3.

Gyroscope data is mandatory if available and should be updated with a frequency of 10 Hz.

4.11.11 ReportHvac

Report HVAC data.

Syntax

```
CinemoError ReportHvac(  
    [in]      int hastargettemperature,  
    [in]      int targettemperatureE3,  
    [in]      int hascurrenttemperature,  
    [in]      int currenttemperatureE3  
) ;
```

Parameters

- **hastargettemperature**

Is the target temperature parameter valid.

- **targettemperatureE3**

Target temperature in degrees Celsius multiplied by 1e3.

- **hascurrenttemperature**

Is the cabin temperature parameter valid.

- **currenttemperatureE3**

Current cabin temperature in degrees Celsius multiplied by 1e3.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

The update frequency for this data is unknown.

4.11.12 ReportLight

Report status of lights.

Syntax

```
CinemoError ReportLight(
    [in]      int hasheadlightstate,
    [in]      CINEMO_ANDROIDAUTO_HEADLIGHT_STATE headlightstate,
    [in]      int hasturnindicatorstate,
    [in]      CINEMO_ANDROIDAUTO_TURNINDICATOR_STATE turnindicatorstate,
    [in]      int hashazardlightson,
    [in]      int hazardlightson
);
```

Parameters

- **hasheadlightstate**

Is the headlightstate parameter valid.

- **headlightstate**

The state of the head lights.

- **hasturnindicatorstate**

Is the turnindicatorstate parameter valid.

- **turnindicatorstate**

The turn indicator state.

- **hashazardlightson**

Is the hazardlightson parameter valid.

- **hazardlightson**

True if hazard lights are on, false otherwise.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

The *headlightstate* field may take one of the following values:

- **CINEMO_ANDROIDAUTO_HEADLIGHT_STATE_OFF**

Head lights off.

- **CINEMO_ANDROIDAUTO_HEADLIGHT_STATE_ON**

Head lights on (day running, low beam).

- **CINEMO_ANDROIDAUTO_HEADLIGHT_STATE_HIGH**

High beams on.

The *turnindicatorstate* field may take one of the following values:

- CINEMO_ANDROIDAUTO_TURNINDICATOR_STATE_NONE
- CINEMO_ANDROIDAUTO_TURNINDICATOR_STATE_LEFT
- CINEMO_ANDROIDAUTO_TURNINDICATOR_STATE_RIGHT

Light status is optional and should be updated on change.

HUMAX CONFIDENTIAL

4.11.13 ReportLocation

Report a GPS location.

Syntax

```
CinemoError ReportLocation(  
    [in]      int latitudeE7,  
    [in]      int longitudeE7,  
    [in]      int hasaccuracy,  
    [in]      int accuracyE3,  
    [in]      int hasaltitude,  
    [in]      int altitudeE2,  
    [in]      int hasspeed,  
    [in]      int speedE3,  
    [in]      int hasbearing,  
    [in]      int bearingE6  
) ;
```

Parameters

- **latitudeE7**

Latitude value [-90.0, +90.0] multiplied by 1e7.

- **longitudeE7**

Longitude value [-180.0, +180.0] multiplied by 1e7.

- **hasaccuracy**

True if accuracy of this location is known and accuracyE3 parameter is valid.

- **accuracyE3**

Horizontal 68% radius meters value multiplied by 1e3.

- **hasaltitude**

True if the altitudeE2 parameter should be considered, false otherwise.

- **altitudeE2**

The altitude in meters above sea level multiplied by 1e2.

- **hasspeed**

True if the speedE3 parameter should be considered, false otherwise.

- **speedE3**

The speed in m/s absolute velocity multiplied by 1e3.

- **hasbearing**

True if the bearingE6 parameter should be considered, false otherwise.

- **bearingE6**

The compass bearing [0, 360) multiplied by 1e6.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Location data is mandatory if available at the head unit. The update frequency should be 1 Hz if a GPS lock is available, otherwise no updates should be sent.

The head unit should send raw GPS values to the Android device with no Dead Reckoning (DR) or Map Matching (MM) applied because the Android device will perform its own DR and MM with the additional sensors and map data it has available.

4.11.14 ReportNightMode

Report value of the day-night sensor.

Syntax

```
CinemoError ReportNightMode(  
    [in]      int nightmode  
) ;
```

Parameters

- **nightmode**

True if night mode is enabled, false otherwise.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Indicates use of night mode when rendering the UI.

Night mode is mandatory and should be updated on change.

4.11.15 ReportOdometer

Report the current odometer value.

Syntax

```
CinemoError ReportOdometer(  
    [in]      int kmsE1,  
    [in]      int hastripkms,  
    [in]      int tripkmsE1  
) ;
```

Parameters

- **kmsE1**

The odometer data in kilometers multiplied by 1e3.

- **hastripkms**

True if the trip kms parameter is valid.

- **tripkmsE1**

Distance traveled since ignition was turned on multiplied by 1e1, in km.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Odometer data is optional and should be updated on change with a resolution of 100 meters.

4.11.16 ReportParkingBrake

Report state of the parking brake.

Syntax

```
CinemoError ReportParkingBrake(  
    [in]      int engaged  
) ;
```

Parameters

- **engaged**

True if the parking brake is engaged, false otherwise.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

The value of this sensor may be used to determine which UI elements can be interacted with.

Parking brake data is optional and should be updated on change.

4.11.17 ReportPassenger

Report passenger presence.

Syntax

```
CinemoError ReportPassenger(  
    [in]      int passengerpresent  
) ;
```

Parameters

- **passengerpresent**

True if a passenger is present, false otherwise.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Google services are a personalized experience, so it is important to know if the passenger seat is occupied before projecting video or audio content that you may not want a passenger to hear.

Passenger data is optional and should be updated on change.

4.11.18 ReportRPM

Report the current RPM of the engine.

Syntax

```
CinemoError ReportRPM(  
    [in]      int rpmE3  
) ;
```

Parameters

- **speedE3**

The engine RPM value multiplied by 1e3.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

RPM data is optional and should be updated with a frequency of 10 Hz.

4.11.19 ReportSpeed

Report the speed of the car.

Syntax

```
CinemoError ReportSpeed(  
    [in]      int speedE3,  
    [in]      int hasCruiseEngaged,  
    [in]      int cruiseEngaged,  
    [in]      int hasCruiseSetSpeed,  
    [in]      int cruiseSetSpeed  
) ;
```

Parameters

- **speedE3**

The speed in m/s signed velocity multiplied by 1e3.

- **hasCruiseEngaged**

True if cruise engaged parameter is valid.

- **cruiseEngaged**

Whether or not cruise control is engaged.

- **hasCruiseSetSpeed**

True if cruise set speed parameter is valid.

- **cruiseSetSpeed**

The speed the cruise control is set at, in m/s.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

The value reported here might be used in dead reckoning the position of the vehicle in the event of a GPS signal loss.

Speed data is mandatory with an update frequency of 10 Hz. Needs to be derived from mechanical rotation at transmission or wheels (not GPS). Must be a negative value when the vehicle is moving in reverse.

4.11.20 ReportTirePressure

Report tire pressure.

Syntax

```
CinemoError ReportTirePressure(  
    [in]      const int * tirepressuresE2,  
    [in]      size_t numentries  
) ;
```

Parameters

- **tirepressuresE2**

An array of tire pressure values in psi multiplied by 1e2.

- **numentries**

The number of elements in the tirepressuresE2 array.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Tire pressure data is optional and should be updated on change.

4.11.21 ReportTollCardData

Report toll card data.

Syntax

```
CinemoError ReportTollCardData(  
    [in]      int isCardPresent  
) ;
```

Parameters

- **isCardPresent**

Current state of the ETC card

Return value

If the method succeeds, it returns CinemoNoError. Otherwise, it returns a CinemoError code.

Remarks

This method is designed to work with the Japanese Electronic Toll Collection (ETC) system.

If isCardPresent is set to 1, the Android UI will display the ETC icon in an enabled state, and if isCardPresent is set to 0, the Android UI will display the ETC icon in a disabled state.

4.11.22 ReportError

Report an error with any of the registered sensors.

Syntax

```
CinemoError ReportError(  
    [in]      CINEMO_ANDROIDAUTO_SENSORTYPE type,  
    [in]      CINEMO_ANDROIDAUTO_SENSOR_ERROR error  
) ;
```

Parameters

- **type**
The type of sensor having a problem.
- **error**
Error code.

Return value

On success, this method returns *CinemoNoError*, otherwise it will return *CinemoErrorArguments*.

Remarks

Report a transient or permanent sensor error to the connected Android device. This is also used to report recovery from such an error state by setting error to OK. Send this message when you cannot recover from an error immediately. If the error is recoverable immediately, there is no need to send this message.

4.12 ICinemoAndroidAutoVendorExtension

The [ICinemoAndroidAutoVendorExtension](#) interface is part of the Cinemo [Android Auto](#) implementation. While the base interface [ICinemoAndroidAuto](#) implements audio and video focus related methods, the [ICinemoAndroidAutoVendorExtension](#) interface is used to send proprietary data to the phone.

This interface needs to be queried from the base [ICinemoAndroidAuto](#) interface object:

```
CinemoAutoPtr<ICinemoAndroidAuto> androidauto;
CinemoAutoPtr<ICinemoAndroidAutoVendorExtension> vendorextension;

CinemoCreateAndroidAuto(androidauto.pp());
androidauto.QueryInterface(vendorextension.pv(), ICinemoAndroidAutoVendorExtension::iid);
```

ICinemoAndroidAutoVendorExtension Methods

METHOD	DESCRIPTION
SendData	Send data

Notes

The Cinemo Android Auto interfaces are in active development, closely following the development of Android Auto by Google. Future changes to this interface are likely.

Remote VEC access

Each Vendor Extension Channel can be accessed from different processes via DDP. When an Android Auto session has been established, the metapool of the corresponding [ICinemoPlayer](#) will contain `CINEMO_METANAME_ANDROIDAUTO_VEC_PATH` entries for each available channel. The value of this entry can be used to open an [ICinemoDataPort](#) instance from the same process or via DDP from any other process. Please note that for DDP connections the path has to be URL-encoded. A short tutorial is available [here](#).

The following restrictions apply:

- A DDP server needs to be running in the process which holds the [ICinemoAndroidAuto](#) instance.
- The VEC callback is mandatory to use this feature.
- If no [ICinemoDataPort](#) interface is attached to a session, all VEC related data will be passed to the [ICinemoAndroidAutoCallbacks::VendorExtensionDataAvailable\(\)](#) callback.
- It is always possible to send data via the [ICinemoAndroidAutoVendorExtension](#) interface, regardless if an [ICinemoDataPort](#) instance is attached to that session or not.
- Dynamically attaching and detaching [ICinemoDataPorts](#) is supported as long as there is only a single instance connected at a time.

4.12.1 SendData

Send data.

Syntax

```
CinemoError SendData(  
    [in] uint32 index,  
    [in] const void * pdata,  
    [in] size_t len  
) ;
```

Parameters

- **index**

The Vendor Extension Channel index.

- **pdata**

The data to be sent.

- **len**

The length of the data to be sent in bytes.

Return value

This method returns *CinemoNoError*.

Remarks

The payload of each vendor extension message is restricted to 512 kB unless that specific extension or message explicitly allows bigger data.

The index corresponds to the Cinemo Metapool title, for which the data should be sent.

4.13 ICinemoAndroidAutoAudioCallbacks

The [ICinemoAndroidAutoAudioCallbacks](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is to provide the interface for the audio sink and source interfaces callbacks of Android Auto which any application must implement.

ICinemoAndroidAutoAudioCallbacks Methods

METHOD	DESCRIPTION
OnAudioSinkFocusRequest	The Android device requests to acquire audio focus
OnAudioSinkStartPlayback	An audio stream is starting
OnAudioSinkStopPlayback	An audio stream is stopping
On AudioSourceMicrophoneRequest	The Android device requests the microphone

4.13.1 OnAudioSinkFocusRequest

The Android device requests to acquire audio focus.

Syntax

```
CinemoError OnAudioSinkFocusRequest(  
    [in]      CINEMO_ANDROIDAUTO_AUDIOFOCUSMODE mode  
) ;
```

Parameters

- **mode**

Requested audio focus mode.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto session requests to acquire audio focus. The head unit has to decide upon this request and respond by calling [ICinemoAndroidAuto::SetAudioFocus\(\)](#).

The mode field may take one of the following values:

- CINEMO_ANDROIDAUTO_AUDIOFOCUSMODE_GAIN
- CINEMO_ANDROIDAUTO_AUDIOFOCUSMODE_GAIN_TRANSIENT
- CINEMO_ANDROIDAUTO_AUDIOFOCUSMODE_GAIN_TRANSIENT_MAY_DUCK
- CINEMO_ANDROIDAUTO_AUDIOFOCUSMODE_RELEASE

4.13.2 OnAudioSinkStartPlayback

An audio stream is starting.

Syntax

```
CinemoError OnAudioSinkStartPlayback(
    [in]      int sessionId,
    [in]      CINEMO_ANDROIDAUTO_AUDIOSTREAM stream
);
```

Parameters

- **sessionId**

The id of the audio session.

- **stream**

The type of the audio stream.

Return value

If this method is successful it should return CinemoNoError. If the audio device is not yet ready it is possible to return CinemoErrorPending in certain special cases.

Remarks

This application defined method will be called whenever the Android Auto session starts playback of an audio stream.

By returning CinemoErrorPending the internal audio processing will be halted until [ICinemoAndroidAuto::ConfirmAudioPlaybackStart\(\)](#) is called for the same stream. This should only be done in very special cases, which are described in detail in the documentation of [ICinemoAndroidAuto::SetAudioFocus\(\)](#).

The stream field may take one of the following values:

- CINEMO_ANDROIDAUTO_AUDIOSTREAM_GUIDANCE
- CINEMO_ANDROIDAUTO_AUDIOSTREAM_MEDIA

4.13.3 OnAudioSinkStopPlayback

An audio stream is stopping.

Syntax

```
CinemoError OnAudioSinkStopPlayback(  
    [in]      int sessionId,  
    [in]      CINEMO_ANDROIDAUTO_AUDIOSTREAM stream  
) ;
```

Parameters

- **sessionId**

The id of the audio session.

- **stream**

The type of the audio stream.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto session stops playback of an audio stream.

4.13.4 On AudioSourceMicrophoneRequest

The Android device requests the microphone.

Syntax

```
int On AudioSourceMicrophoneRequest (
    [in]      int openMicrophone,
    [in]      int ancEnabled,
    [in]      int ecEnabled
);
```

Parameters

- **openMicrophone**

True if the microphone should be opened, false if it should be closed.

- **ancEnabled**

True if active noise cancellation should be enabled, false otherwise.

- **ecEnabled**

True if echo cancellation should be enabled, false otherwise.

Return value

This method should return *CINEMO_ANDROIDAUTO_STATUS_SUCCESS* or a positive number on success, which will then be used as the session id. Any negative *CINEMO_ANDROIDAUTO_STATUS* enum value can be used to indicate an error. When a session is closed the return value is ignored.

Remarks

This application defined method will be called before the Android Auto session opens a microphone session or after an existing microphone session has been closed.

4.14 ICinemoAndroidAutoBluetoothCallbacks

The [ICinemoAndroidAutoBluetoothCallbacks](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is to provide the interface for the bluetooth interface callbacks of Android Auto which an application may implement. If this interface is not implemented on the client's side, then the corresponding endpoint will not be registered.

ICinemoAndroidAutoBluetoothCallbacks Methods

METHOD	DESCRIPTION
OnBluetoothAuthenticationResult	Bluetooth authentication result notification
OnBluetoothPairingRequest	Bluetooth pairing is requested

4.14.1 OnBluetoothAuthenticationResult

Bluetooth authentication result notification.

Syntax

```
CinemoError OnBluetoothAuthenticationResult(  
    [in]      CINEMO_ANDROIDAUTO_STATUS status  
) ;
```

Parameters

- **status**

The authentication status, as defined in enum CINEMO_ANDROIDAUTO_STATUS

Return value

The return value of this callback is ignored.

Remarks

This application defined function will be called whenever the Android device wants to notify the head unit about a Bluetooth authentication result.

Once this function is called, the head unit must accept or reject the authentication data for the current Bluetooth pairing request.

Whenever the head unit has called [ICinemoAndroidAutoBluetooth::SendAuthData\(\)](#), it must not complete the pairing authentication until this callback function has been called. This is necessary to ensure that the head unit has paired with the correct mobile device.

Successful authentication is indicated when a value of CINEMO_ANDROIDAUTO_STATUS_SUCCESS is passed to this function, and CINEMO_ANDROIDAUTO_STATUS_BLUETOOTH_INVALID_AUTH_DATA or CINEMO_ANDROIDAUTO_STATUS_BLUETOOTH_AUTH_DATA_MISMATCH indicate common failure modes. Any status other than CINEMO_ANDROIDAUTO_STATUS_SUCCESS must be treated as an authentication failure.

Any authentication result received within 60 seconds from the time the head unit has called the [ICinemoAndroidAutoBluetooth::OnReadyForPairing\(\)](#) function shall be handled normally by this function. When 60 seconds have passed, the head unit must discard the authentication result, treating it as though the mobile device has reported the authentication has failed.

4.14.2 OnBluetoothPairingRequest

Bluetooth pairing is requested.

Syntax

```
CinemoError OnBluetoothPairingRequest (
    [in]      const char * szphoneaddress,
    [in]      CINEMO_ANDROIDAUTO_BLUETOOTHPAIRING pairingMethod
);
```

Parameters

- **szphoneaddress**

The Bluetooth MAC address of the Android device

- **pairingMethod**

The requested Bluetooth pairing method

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android device wants to start the Bluetooth pairing process with the head unit.

Once this function is called, the head unit must make itself prepared for Bluetooth pairing which will be initiated by the phone.

To be ready for pairing, the head unit must disconnect all other HFP (Hands-Free Profile) connections, set the Android phone's priority as the highest and make the car's Bluetooth module discoverable to the phone.

When the head unit is ready to be paired, it must call [ICinemoAndroidAutoBluetooth::OnReadyForPairing\(\)](#). Then the phone will initiate the Bluetooth pairing-and-connection process with the head unit. In this process the pairing step will be skipped if both the phone and the car have link keys (pairing information) for each other. In that case the phone will initiate a HFP (Hands-Free Profile) connection only.

The head unit must be prepared for pairing and call [ICinemoAndroidAutoBluetooth::OnReadyForPairing\(\)](#), even if it has already been paired with the phone. This is necessary because the phone may have lost the pairing information, even though the head unit has not. In that case, the phone may initiate pairing.

If the head unit cannot be prepared for pairing within 1 second of receiving the [BluetoothPairingRequest\(\)](#) callback, the head unit should call [ICinemoAndroidAutoBluetooth::RequestDelayedPairing\(\)](#). The phone will then assume that the head unit will call [ICinemoAndroidAutoBluetooth::OnReadyForPairing\(\)](#) after a while.

Even if the Android device and the head unit have already been paired after the head unit calls [ICinemoAndroidAutoBluetooth::OnReadyForPairing\(\)](#), the phone may unpair and invoke

[BluetoothPairingRequest\(\)](#) again, in order to fix a corrupted state. The head unit must be able to handle BluetoothPairingRequest messages multiple times in the same Android Auto session.

The pairingMethod field may take one of the following values:

- CINEMO_ANDROIDAUTO_BLUETOOTHPAIRING_OOB
- CINEMO_ANDROIDAUTO_BLUETOOTHPAIRING_NUMERIC_COMPARISON
- CINEMO_ANDROIDAUTO_BLUETOOTHPAIRING_PASSKEY_ENTRY
- CINEMO_ANDROIDAUTO_BLUETOOTHPAIRING_PIN

4.15 ICinemoAndroidAutoCoreCallbacks

The [ICinemoAndroidAutoCoreCallbacks](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is to provide the interface for the core Android Auto callbacks which any application must implement.

ICinemoAndroidAutoCoreCallbacks Methods

METHOD	DESCRIPTION
OnBatteryStatusNotification	A battery status notification has been received
OnBeginSession	An Android Auto session has started
OnBugreportRequest	Indicates that the MD wants the HU to capture a bugreport
OnEndSession	An Android Auto session has ended
OnNewConnection	A connection request for a new session has arrived
OnServiceDiscoveryRequest	A service discovery request has arrived
OnSessionEvent	An event for an active session has arrived
OnNavigationFocusRequest	The Android device requests to acquire navigation focus
OnUnrecoverableError	Called when an unrecoverable error has been encountered
OnVoiceSessionNotification	An Android Auto session starts or ends a voice session

4.15.1 OnBatteryStatusNotification

A battery status notification has been received.

Syntax

```
CinemoError OnBatteryStatusNotification (
    [in]      int batteryLevel,
    [in]      int timeRemainingS,
    [in]      int batteryCritical
);
```

Parameters

- **batteryLevel**

The battery level of the device.

- **timeRemainingS**

The remaining time in seconds.

- **batteryCritical**

Indicator if the battery level is critical.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever a battery status notification is received from the connected device.

4.15.2 OnBeginSession

An Android Auto session has started.

Syntax

```
CinemoError OnBeginSession();
```

Return value

To accept this session return CinemoNoError – every other error code will abort the session initialization.

Remarks

This application defined function will be called whenever an Android Auto session has been established with the Android device. If the application does NOT wish a session to be established, it should return an error code other than CinemoNoError.

4.15.3 OnBugreportRequest

Indicates that the MD wants the HU to capture a bugreport.

Syntax

```
CinemoError OnBugreportRequest(  
    [in]      sint64 timestamp  
) ;
```

Parameters

- **timestamp**

The (remote) timestamp of the request.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the mobile device wants the head unit to capture a bugreport. The head unit can choose to ignore this if it is sent with high frequency – it is the head unit's responsibility to ensure that the system cannot be DoS'd.

4.15.4 OnEndSession

An Android Auto session has ended.

Syntax

```
CinemoError OnEndSession(  
    [in]      CINEMO_ANDROIDAUTO_SESSION_END_REASON reason  
) ;
```

Parameters

- **reason**

Reason why the session has been terminated.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto session has been terminated. This might be the case if the USB cable is disconnected or the Android Auto Projection Application on the phone is terminated.

4.15.5 OnNewConnection

A connection request for a new session has arrived.

Syntax

```
CinemoError OnNewConnection(  
    [in]      const char * szurl  
) ;
```

Parameters

- **szurl**

The URL of the connecting device (WiFi or USB)

Return value

The return value will determine if the new WiFi connection will be accepted (CinemoNoError) or not (any other value). For the USB case the session will be accepted regardless of the return value.

Remarks

This application defined method will be called whenever a new client connects to the Android Auto socket or a session is started with an USB device.

If this is an expected WiFi connection CinemoNoError can be returned. In this case Cinemo will internally switch the active session to this new device and terminate the existing connection according to the Google specification.

If this is an unexpected WiFi connection any CinemoError code can be returned. This will reject the new connection and keep an existing Android Auto session running.

For the USB case the return value has no significance and is not analyzed. Therefore, this callback should be used for informational purposes only.

4.15.6 OnServiceDiscoveryRequest

A service discovery request has arrived.

Syntax

```
CinemoError OnServiceDiscoveryRequest(  
    [in]      const uchar * smallIcon,  
    [in]      size_t smallIcon_len,  
    [in]      const uchar * mediumIcon,  
    [in]      size_t mediumIcon_len,  
    [in]      const uchar * largeIcon,  
    [in]      size_t largeIcon_len,  
    [in]      const char * szlabel,  
    [in]      const char * szdeviceName,  
    [in]      const char * szinstanceId  
) ;
```

Parameters

- **smallIcon**

Data for 32x32 png image

- **smallIcon_len**

Length of smallIcon data

- **mediumIcon**

Data for 64x64 png image

- **mediumIcon_len**

Length of mediumIcon data

- **largeIcon**

Data for 128x128 png image

- **largeIcon_len**

Length of largeIcon data

- **szlabel**

A label that may be displayed alongside the icon

- **szdeviceName**

A friendly device name

- **szinstanceId**

A unique mobile device ID which is stable per an Android Auto installation

Return value

The return value of this callback is ignored.

Remarks

This application defined function will be called whenever the Android Auto session has been established by the Android device. This call sends across an icon set and a label that can be used by the native UI to display a button that allows users to switch back to projected mode.

For USB connections it is possible to block this callback for an extended period of time in order to wait for user input. A session may then be rejected using the return value of the [ICinemoAndroidAutoCoreCallbacks::OnBeginSession](#) callback, which is guaranteed to be called after this callback.

4.15.7 OnSessionEvent

An event for an active session has arrived.

Syntax

```
CinemoError OnSessionEvent (
    [in]      CINEMO_ANDROIDAUTO_EVENT event
);
```

Parameters

- **event**
CINEMO_ANDROIDAUTO_EVENT enum

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever an event for an active session becomes available.

4.15.8 OnNavigationFocusRequest

The Android device requests to acquire navigation focus.

Syntax

```
CinemoError OnNavigationFocusRequest (
    [in]      CINEMO_ANDROIDAUTO_NAVIGATIONFOCUSMODE mode
);
```

Parameters

- **mode**

Requested navigation focus mode

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto session requests to acquire navigation focus. The head unit has to decide upon this request and respond by calling ICinemoAndroidAuto::SetNavigationFocus().

4.15.9 OnUnrecoverableError

Called when an unrecoverable error has been encountered.

Syntax

```
CinemoError OnUnrecoverableError(  
    [in]      CINEMO_ANDROIDAUTO_ERROR error  
) ;
```

Parameters

- **error**
CINEMO_ANDROIDAUTO_ERROR type.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever an unrecoverable error has been encountered. It is recommended to reset the USB link at this point. A re-establishment of the Android Auto connection may be attempted afterwards.

4.15.10 OnVoiceSessionNotification

An Android Auto session starts or ends a voice session.

Syntax

```
CinemoError OnVoiceSessionNotification(  
    [in]      CINEMO_ANDROIDAUTO_VOICESESSION_STATUS status  
) ;
```

Parameters

- **status**

The status of the voice recognition session.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto session sends a notification for a voice recognition session.

The status field may take one of the following values:

- CINEMO_ANDROIDAUTO_VOICESESSION_STATUS_START
- CINEMO_ANDROIDAUTO_VOICESESSION_STATUS_END

4.16 ICinemoAndroidAutoInputCallbacks

The [ICinemoAndroidAutoInputCallbacks](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is to provide the interface for the input interface callbacks of Android Auto which an application may implement. If this interface is not implemented on the client's side, then the corresponding endpoint will not be registered.

ICinemoAndroidAutoInputCallbacks Methods

METHOD	DESCRIPTION
OnInputFeedback	Input feedback message received

4.16.1 OnInputFeedback

Input feedback message received.

Syntax

```
CinemoError OnInputFeedback(  
    [in]      int display_index,  
    [in]      CINEMO_ANDROIDAUTO_INPUT_FEEDBACK_EVENT event  
) ;
```

Parameters

- **display_index**

The metapool index of the display which received this request

- **event**

The input feedback event

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever input feedback is received from the connected device.

4.17 ICinemoAndroidAutoMediaBrowserCallbacks

The [ICinemoAndroidAutoMediaBrowserCallbacks](#) interface is part of the Cinemo Android Auto implementation. Its main purpose is to provide the interface for the media browser interface callbacks of Android Auto which an application may implement. If this interface is not implemented on the client's side, then the corresponding endpoint will not be registered.

ICinemoAndroidAutoMediaBrowserCallbacks Methods

METHOD	DESCRIPTION
OnMediaBrowserListNode	Media metadata for a list node has arrived
OnMediaBrowserRootNode	Media metadata for a root node has arrived
OnMediaBrowserSongNode	Media metadata for a song node has arrived
OnMediaBrowserSourceNode	Media metadata for a source node has arrived

4.17.1 OnMediaBrowserListNode

Media metadata for a list node has arrived.

Syntax

```
CinemoError OnMediaBrowserListNode(
    [in]      const CinemoAndroidAutoMediaBrowserListNode & node
);
```

Parameters

- **node**

MediaBrowser list node structure.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called as a response to [ICinemoAndroidAutoMediaBrowser::GetNode\(\)](#).

The **CinemoAndroidAutoMediaBrowserListNode** structure is defined as follows:

```
typedef struct {
    CinemoAndroidAutoMediaBrowserList media_list;
    int start;
    int total;
    CinemoAndroidAutoMediaBrowserSong * songs;
    size_t num_songs;
} CinemoAndroidAutoMediaBrowserListNode;
```

The **CinemoAndroidAutoMediaBrowserSong** structure is defined as follows:

```
typedef struct {
    const char * szpath;
    const char * szname;
    const char * szartist;
    const char * szalbum;
} CinemoAndroidAutoMediaBrowserSong;
```

4.17.2 OnMediaBrowserRootNode

Media metadata for a root node has arrived.

Syntax

```
CinemoError OnMediaBrowserRootNode(
    [in]      const CinemoAndroidAutoMediaBrowserRootNode & node
);
```

Parameters

- **node**

MediaBrowser root node structure.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called as a response to [ICinemoAndroidAutoMediaBrowser::GetNode\(\)](#).

The `CinemoAndroidAutoMediaBrowserRootNode` structure is defined as follows:

```
typedef struct {
    const char * szpath;
    CinemoAndroidAutoMediaBrowserSource * sources;
    size_t num_sources;
} CinemoAndroidAutoMediaBrowserRootNode;
```

The `CinemoAndroidAutoMediaBrowserSource` structure is defined as follows:

```
typedef struct {
    const char * szpath;
    const char * szname;
    const uchar * album_art;
    size_t album_art_len;
} CinemoAndroidAutoMediaBrowserSource;
```

4.17.3 OnMediaBrowserSongNode

Media metadata for a song node has arrived.

Syntax

```
CinemoError OnMediaBrowserSongNode(
    [in]      const CinemoAndroidAutoMediaBrowserSongNode & node
);
```

Parameters

- **node**

MediaBrowser song node structure.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called as a response to [ICinemoAndroidAutoMediaBrowser::GetNode\(\)](#).

The `CinemoAndroidAutoMediaBrowserSongNode` structure is defined as follows:

```
typedef struct {
    CinemoAndroidAutoMediaBrowserSong song;
    const uchar * album_art;
    size_t album_art_len;
    int duration_seconds;
} CinemoAndroidAutoMediaBrowserSongNode;
```

`album_art` is a PNG of art for this song (if requested) with a maximum resolution of 256x256.

The `CinemoAndroidAutoMediaBrowserSong` structure is defined as follows:

```
typedef struct {
    const char * szpath;
    const char * szname;
    const char * szartist;
    const char * szalbum;
} CinemoAndroidAutoMediaBrowserSong;
```

4.17.4 OnMediaBrowserSourceNode

Media metadata for a source node has arrived.

Syntax

```
CinemoError OnMediaBrowserSourceNode(
    [in]      const CinemoAndroidAutoMediaBrowserSourceNode & node
);
```

Parameters

- **node**

MediaBrowser source node structure.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called as a response to [ICinemoAndroidAutoMediaBrowser::GetNode\(\)](#).

The `CinemoAndroidAutoMediaBrowserSourceNode` structure is defined as follows:

```
typedef struct {
    CinemoAndroidAutoMediaBrowserSource source;
    int start;
    int total;
    CinemoAndroidAutoMediaBrowserList * lists;
    size_t num_lists;
} CinemoAndroidAutoMediaBrowserSourceNode;
```

The `CinemoAndroidAutoMediaBrowserList` structure is defined as follows:

```
typedef struct {
    const char * szpath;
    CINEMO_ANDROIDAUTO_MEDIALIST_TYPE type;
    const char * szname;
    const uchar * album_art;
    size_t album_art_len;
} CinemoAndroidAutoMediaBrowserList;
```

`album_art` is a PNG of art for this media source (if requested) with a maximum resolution of 256x256.

The `type` field may take one of the following values:

- `CINEMO_ANDROIDAUTO_MEDIALIST_TYPE_PLAYLIST`
- `CINEMO_ANDROIDAUTO_MEDIALIST_TYPE_ALBUM`
- `CINEMO_ANDROIDAUTO_MEDIALIST_TYPE_ARTIST`
- `CINEMO_ANDROIDAUTO_MEDIALIST_TYPE_STATION`

- CINEMO_ANDROIDAUTO_MEDIALIST_TYPE_GENRE

HUMAX Confidential Automotive

4.18 ICinemoAndroidAutoMediaPlaybackCallbacks

The [ICinemoAndroidAutoMediaPlaybackCallbacks](#) interface is part of the Cinemo Android Auto implementation. Its main purpose is to provide the interface for the audio sink and source interfaces callbacks of Android Auto which any application must implement.

ICinemoAndroidAutoMediaPlaybackCallbacks Methods

METHOD	DESCRIPTION
OnMediaPlaybackMetadata	MediaPlayback metadata has arrived
OnMediaPlaybackStatus	MediaPlayback status update has arrived

4.18.1 OnMediaPlaybackMetadata

MediaPlayback metadata has arrived.

Syntax

```
CinemoError OnMediaPlaybackMetadata(  
    [in]      const CinemoAndroidAutoMediaPlaybackMetadata & metadata  
) ;
```

Parameters

- **metadata**

Metadata for the current playback.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto MediaPlayback metadata changes.

The CinemoAndroidAutoMediaPlaybackMetadata structure is defined as follows:

```
typedef struct {  
    const char * szsong;  
    const char * szalbum;  
    const char * szartist;  
    const uchar * album_art;  
    size_t album_art_len;  
    const char * szplaylist;  
    int duration_seconds;  
    int rating;  
} CinemoAndroidAutoMediaPlaybackMetadata;
```

album_art is encoded as PNG with an image resolution up to 256x256.

4.18.2 OnMediaPlaybackStatus

MediaPlayback status update has arrived.

Syntax

```
CinemoError OnMediaPlaybackStatus(
    [in]      const CinemoAndroidAutoMediaPlaybackStatus & status
);
```

Parameters

- **status**

MediaPlayback status data.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto MediaPlayback status updates.

The `CinemoAndroidAutoMediaPlaybackStatus` structure is defined as follows:

```
typedef struct {
    CINEMO_ANDROIDAUTO_MEDIAPLAYBACKSTATUS_STATE state;
    const char * szmedia_source;
    int playback_seconds;
    int shuffle;
    int repeat;
    int repeat_one;
} CinemoAndroidAutoMediaPlaybackStatus;
```

where the `state` field may take one of the following values:

- `CINEMO_ANDROIDAUTO_MEDIAPLAYBACKSTATUS_STATE_STOPPED`
- `CINEMO_ANDROIDAUTO_MEDIAPLAYBACKSTATUS_STATE_PLAYING`
- `CINEMO_ANDROIDAUTO_MEDIAPLAYBACKSTATUS_STATE_PAUSED`.

4.19 ICinemoAndroidAutoNavigationCallbacks

The [ICinemoAndroidAutoNavigationCallbacks](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is to provide a callbacks interface for the navigation interface of Android Auto which an application may implement. If this interface is not implemented on the client's side and is set to `nullptr` on [ICinemoAndroidAuto::SetCallbacks](#) method call, then the corresponding endpoint will not be registered.

ICinemoAndroidAutoNavigationCallbacks Methods

METHOD	DESCRIPTION
OnNavigationCurrentPosition	New navigation data about the current position has arrived
OnNavigationState	New navigation state data has arrived
OnNavigationStatus	The Android Auto navigation status has changed

4.19.1 OnNavigationCurrentPosition

New navigation data about the current position has arrived.

Syntax

```
CinemoError OnNavigationCurrentPosition (
    [in]      const CinemoAndroidAutoNavigationCurrentPosition & position
);
```

Parameters

- **position**

Structure containing the information relevant to the current position

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto navigation sends a new message about current position.

The content of the CinemoAndroidAutoNavigationCurrentPosition structure closely follows the content of the structures, provided by the Google receiver library. For more information please refer to the Google documentation.

4.19.2 OnNavigationState

New navigation state data has arrived.

Syntax

```
CinemoError OnNavigationState(  
    [in]      const CinemoAndroidAutoNavigationState & state  
) ;
```

Parameters

- **state**

Structure containing the navigation state information

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto navigation sends a new state message.

The content of the CinemoAndroidAutoNavigationState structure closely follows the content of the structures provided by the Google receiver library. For more information please refer to the Google documentation.

4.19.3 OnNavigationStatus

The Android Auto navigation status has changed.

Syntax

```
CinemoError CinemoAndroidAuto_NavigationStatus(  
    [in]      CINEMO_ANDROIDAUTO_NAVIGATION_STATUS status  
) ;
```

Parameters

- **status**

Navigation status.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto navigation status changes.

The status field can assume one of the following values:

- CINEMO_ANDROIDAUTO_NAVIGATION_STATUS_UNAVAILABLE
- CINEMO_ANDROIDAUTO_NAVIGATION_STATUS_ACTIVE
- CINEMO_ANDROIDAUTO_NAVIGATION_STATUS_INACTIVE

4.20 ICinemoAndroidAutoNotificationCallbacks

The [ICinemoAndroidAutoNotificationCallbacks](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is to provide a callbacks interface for the notification interface of Android Auto which an application may implement. If this interface is not implemented on the client's side and is set to `nullptr` on [ICinemoAndroidAuto::SetCallbacks](#) method call, then the corresponding endpoint will not be registered.

ICinemoAndroidAutoNotificationCallbacks Methods

METHOD	DESCRIPTION
OnNotificationAck	A notification acknowledgement was received
OnNotificationArrival	A notification from the Android device has arrived
OnNotificationSubscriptionStatus	Phone subscription to the notification service has changed

4.20.1 OnNotificationAck

A notification acknowledgement was received.

Syntax

```
CinemoError OnNotificationAck(  
    [in]      const char * szid  
) ;
```

Parameters

- **szid**

The id of the notification that was acknowledged.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android device acknowledges a notification sent by the head unit. This will only be called for notifications that explicitly request an acknowledgment.

4.20.2 OnNotificationArrival

A notification from the Android device has arrived.

Syntax

```
CinemoError OnNotificationArrival(  
    [in]      const char * sztext,  
    [in]      int hasId,  
    [in]      const char * id,  
    [in]      int hasIcon,  
    [in]      const uchar * picon,  
    [in]      size_t icon_len  
) ;
```

Parameters

- **sztext**

The text of the notification to be displayed.

- **hasId**

If true, the id parameter contains valid data.

- **id**

An identifier for this notification.

- **hasIcon**

If true, the icon pointer contains valid data.

- **picon**

A pointer to the icon data (PNG format), maximum resolution is 256x256.

- **icon_len**

The size of the icon in bytes.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever a notification from the Android device arrives.

If an id is present, you must call [ICinemoAndroidAutoNotification::Ack\(\)](#) with the id once the notification has been handled.

4.20.3 OnNotificationSubscriptionStatus

Phone subscription to the notification service has changed.

Syntax

```
CinemoError OnNotificationSubscriptionStatus(  
    [in]      int subscribed  
) ;
```

Parameters

- **subscribed**

Subscription status.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the status of the phones subscription to the notification service changes. If you try to send a notification before the phone is subscribed, it will fail.

4.21 ICinemoAndroidAutoPhoneCallbacks

The [ICinemoAndroidAutoPhoneCallbacks](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is to provide a callbacks interface for the phone interface of Android Auto which an application may implement. If this interface is not implemented on the client's side and is set to `NULL` on [ICinemoAndroidAuto::SetCallbacks](#) method call, then the corresponding endpoint will not be registered.

ICinemoAndroidAutoPhoneCallbacks Methods

METHOD	DESCRIPTION
OnPhoneStatus	A phone status message was received

4.21.1 OnPhoneStatus

A phone status message was received.

Syntax

```
CinemoError OnPhoneStatus(
    [in]      const CinemoAndroidAutoPhoneStatus & status
);
```

Parameters

- **status**

Phone status message.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever an Android Auto PhoneStatus message is received.

The CinemoAndroidAutoPhoneStatus structure is defined as follows:

```
typedef struct {
    CinemoAndroidAutoPhoneCall * calls;
    size_t num_calls;
    int signal_strength;
} CinemoAndroidAutoPhoneStatus;
```

The CinemoAndroidAutoPhoneCall structure is defined as follows:

```
typedef struct {
    CINEMO_ANDROIDAUTO_PHONECALL_STATE state;
    const char * szcaller_number;
    const char * szcaller_id;
    const char * szcaller_number_type;
    int call_duration_seconds;
    const uchar * caller_thumbnail;
    size_t caller_thumbnail_len;
} CinemoAndroidAutoPhoneCall;
```

where state may assume one of the following values:

- CINEMO_ANDROIDAUTO_PHONECALL_STATE_IN_CALL
- CINEMO_ANDROIDAUTO_PHONECALL_STATE_ON_HOLD
- CINEMO_ANDROIDAUTO_PHONECALL_STATE_INACTIVE
- CINEMO_ANDROIDAUTO_PHONECALL_STATE_INCOMING
- CINEMO_ANDROIDAUTO_PHONECALL_STATE_CONFERENCED

- CINEMO_ANDROIDAUTO_PHONECALL_STATE_MUTED

and `caller_thumbnail` is a PNG image of the caller (if available) with a maximum resolution of 256x256.

HUMAX Confidential

4.22 ICinemoAndroidAutoVendorExtensionCallbacks

The [ICinemoAndroidAutoVendorExtensionCallbacks](#) interface is part of the Cinemo Android Auto implementation. Its main purpose is to provide a callbacks interface for the vendor extension interface of Android Auto which an application may implement. If this interface is not implemented on the client's side and is set to `nullptr` on [ICinemoAndroidAuto::SetCallbacks](#) method call, then the corresponding endpoint will not be registered.

ICinemoAndroidAutoVendorExtensionCallbacks Methods

METHOD	DESCRIPTION
OnVendorExtensionDataAvailable	A new message has arrived on the VendorExtension channel

4.22.1 OnVendorExtensionDataAvailable

A new message has arrived on the VendorExtension channel.

Syntax

```
CinemoError OnVendorExtensionDataAvailable(  
    [in]      uint32 index,  
    [in]      const uchar * pdata,  
    [in]      size_t data_len  
) ;
```

Parameters

- **index**

The Vendor Extension Channel index.

- **pdata**

A pointer to the data.

- **data_len**

The size of the data.

Return value

If this method is successful it should return *CinemoNoError*.

Remarks

This application defined method will be called whenever a new message arrives on the vendor extension channel on the Android Auto receiver library. The index corresponds to the Cinemo Metapool title, for which the mobile device sent data.

4.23 ICinemoAndroidAutoVideoCallbacks

The [ICinemoAndroidAutoVideoCallbacks](#) interface is part of the Cinemo [Android Auto](#) implementation. Its main purpose is to provide the interface for the video interface callbacks of Android Auto which any application must implement.

ICinemoAndroidAutoVideoCallbacks Methods

METHOD	DESCRIPTION
OnVideoConfig	The selected video configuration has been reported
OnVideoFocusRequest	The Android device requests to acquire video focus
OnVideoReady	The video encoder on the Android device is ready
OnVideoUiConfigChange	The UI configuration of a specific display has changed

4.23.1 OnVideoConfig

The selected video configuration has been reported by the Android device.

Syntax

```
CinemoError OnVideoConfig(  
    [in]      uint32 display_index,  
    [in]      uint32 config_id  
) ;
```

Parameters

- **display_index**

The metapool index of the display for which this configuration is reported

- **config_id**

The video configuration ID.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called if the selected video configuration is reported by the Android device. The configuration ID corresponds to the index in the metapool, provided to [CinemoAndroidAuto::Setup\(\)](#).

4.23.2 OnVideoFocusRequest

The Android device requests to acquire video focus.

Syntax

```
CinemoError OnVideoFocusRequest(
    [in]      uint32 display_index,
    [in]      CINEMO_ANDROIDAUTO_VIDEOFOCUSMODE mode,
    [in]      CINEMO_ANDROIDAUTO_VIDEOFOCUSREASON reason
);
```

Parameters

- **display_index**

The metapool index of the display which received this request

- **mode**

The focus mode request by the Android device.

- **reason**

The reason for this video focus request.

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the Android Auto session requests to acquire video focus. The head unit is expected to answer this request by calling [ICinemoAndroidAutoDisplay::SetVideoFocus\(\)](#) of the [ICinemoAndroidAutoDisplay](#) object corresponding to the `display_index` parameter.

The `mode` field may take one of the following values:

- CINEMO_ANDROIDAUTO_VIDEOFOCUSMODE_UNKNOWN
- CINEMO_ANDROIDAUTO_VIDEOFOCUSMODE_PROJECTED
- CINEMO_ANDROIDAUTO_VIDEOFOCUSMODE_NATIVE
- CINEMO_ANDROIDAUTO_VIDEOFOCUSMODE_NATIVE_TRANSIENT

The `reason` field may take one of the following values:

- CINEMO_ANDROIDAUTO_VIDEOFOCUSREASON_UNKNOWN
- CINEMO_ANDROIDAUTO_VIDEOFOCUSREASON_PHONE_SCREEN_OFF
- CINEMO_ANDROIDAUTO_VIDEOFOCUSREASON_LAUNCH_NATIVE

4.23.3 OnVideoReady

The video encoder on the Android device is ready.

Syntax

```
CinemoError OnVideoReady(  
    [in]      uint32 display_index  
) ;
```

Parameters

- **display_index**

The metapool index of the display for which this configuration is reported

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the video encoder on the Android device is ready. From this time on the head unit may request video focus.

4.23.4 OnVideoUiConfigChange

The UI configuration of a specific display has changed.

Syntax

```
CinemoError OnVideoUiConfigChange (
    [in]      uint32 display_index,
    [in]      int has_margins,
    [in]      const CinemoAndroidAutoInsets & margins,
    [in]      int has_insets,
    [in]      const CinemoAndroidAutoInsets & insets,
    [in]      int has_stable_insets,
    [in]      const CinemoAndroidAutoInsets & stable_insets,
    [in]      int has_theme,
    [in]      CINEMO_ANDROIDAUTO_UI_THEME ui_theme
);
```

Parameters

- **display_index**

The metapool index of the display whose configuration has changed

- **has_margins**

Non-zero if new margin values are available

- **margins**

If has_margins is set, contains the updated margins

- **has_insets**

Non-zero if new inset values are available

- **insets**

If has_insets is set, contains the updated insets

- **has_stable_insets**

Non-zero if new stable_inset values are available

- **stable_insets**

If has_stable_insets is set, contains the updated stable_margins

- **has_theme**

Non-zero if a new UI theme value is available

- **ui_theme**

If has_theme is set, contains the active UI theme

Return value

The return value of this callback is ignored.

Remarks

This application defined method will be called whenever the video sink receives a UI configuration update from the mobile device.

All current UI configuration parameters can also be retrieved from the [ICinemoAndroidAutoDisplay](#) object via the corresponding access methods.

5. Cinemo Interfaces for Apple Devices

All Cinemo interfaces are defined in C++ header files which are provided with the Cinemo Media Engine SDK. These headers are pre-processed based on the enabled feature set for a project, and so may not include all the interfaces and API functions documented here. For example, [ICinemoMM](#) interface is only available when Cinemo Media Management feature is enabled. If you require an interface or function which is described in this document but is missing from the Cinemo SDK headers, then please contact your Cinemo representative to provide you an updated SDK with these features enabled.

All Cinemo interfaces are defined in C++ and follow the Component Object Model (COM) specification. As defined by COM, all Cinemo interfaces derive from a standard [ICinemoUnknown](#) interface, which consists of three methods:

- [ICinemoUnknown::AddRef\(\)](#) and [ICinemoUnknown::Release\(\)](#), reference counting to control the lifetime of interfaces.
- [ICinemoUnknown::QueryInterface\(\)](#), access to interfaces by ID.

All Cinemo interfaces are reflexive, symmetric, and transitive. The reflexive property means that calling [ICinemoUnknown::QueryInterface\(\)](#) on a given interface with the interface's ID will return the same instance of the interface. The symmetric property means that when interface B is retrieved from interface A by calling [ICinemoUnknown::QueryInterface\(\)](#), interface A is also retrievable from interface B. The transitive property means that if interface B is obtainable from interface A and interface C is obtainable from interface B, then interface C is also retrievable from interface A.

5.1 General remarks for Apple devices

Cinemo provides support for Apple devices via the Cinemo Media Engine abstraction. From Cinemo Media Engines point of view an Apple device can be used to play back the media present on the Apple device. Cinemo strongly recommends the usage of [ICinemoPlayer2](#) when doing media playback with an Apple device. The now legacy [ICinemoPlayer](#) lacks some functionality, which is mandatory to be able to pass Apples certification for iAP playback.

The additional features (not necessary related to media playback) of an Apple Device can be accessed via the [ICinemolAP](#) interface. If more than media play back shall be performed with an attached Apple device, then the identification of the Cinemo Media Engine towards the Apple device must be customized. The customization of the identification process can be performed via the [ICinemolAP](#) interface.

Cinemo supports both USB (host and device mode) and wireless (e.g. Bluetooth) connections to the Apple device.

An instance of [ICinemolAP](#) is only a handle to the connection which will be established to the Apple device in Cinemo Media Engine. Thus it is possible to create multiple instances of [ICinemolAP](#) where all instances communicate with the same Apple device. It is important to understand that the very first contact to the Apple device will determine the characteristics of the connection to the Apple device in terms of the required identification process. Subsequent instances of [ICinemolAP](#) can not change such characteristics, while the same connection is still alive. A change is only possible if the life cycle of the connection is controlled by the user of the Cinemo API.

5.1.1 Control of life cycle for a connection to an Apple device

Prior to using an Apple device its attachment to the system must be detected. Cinemo has support for device detection for certain protocols via the [ICinemoVFS](#) interface. But in many cases the device detection is performed by the underlying operating system and as a result an Apple device must be connected to Cinemo Media Engine via the provided Cinemo API.

To control the life cycle of such a connection within the Cinemo Media Engine the following procedure is recommended:

1. Before using any other Cinemo API with the newly detected Apple device perform the identification and authentication process via [ICinemolAP::Open\(\)](#) method. It is of critical importance to pass to this first call to [ICinemolAP::Open\(\)](#) the CINEMO_IAP_OPEN_DEVICE flag. In order to omit certain possible problems with the nature of the Apple devices it is also recommended to register any callback interfaces at this point. More information about callback handling can be found in a later section of this chapter.
2. Keep this instance of [ICinemolAP](#) alive as long other Cinemo APIs are used with the Apple device.
3. If all other uses of the Apple device are finished (and all Cinemo API instances referring to that Apple device have been closed or released) then the initial instance of [ICinemolAP](#) must be closed by calling [ICinemolAP::Close\(\)](#) and the instance must be released. Only this will close the communication channel to the Apple device too. After this point the whole connection and usage procedure can start again.

This procedure should be followed even in the case the Apple device was physically disconnected (e.g. from the USB bus or the Bluetooth connection has been closed on the Apple device etc.).

5.1.2 IAP identification with ICinemolAP

The [ICinemolAP](#) interface allows custom identification of the accessory with the Apple device. This is because Cinemo cannot know in advance which features of the Apple device will be required by the application. The identification is possible only if the Apple device has not yet been opened with [ICinemoPlaylist](#) or [ICinemoVFS](#) previously, because both interfaces will perform a Cinemo specific identification when the connection to the Apple device was not yet established. In this case the [ICinemolAP](#) interface can be used to access other provided functionality – if enabled with the Cinemo internal accessory identification – but a custom identification is no longer possible.

When obtaining the [ICinemolAP](#) interface by means of [CinemoCreateIAP\(\)](#) before any other Cinemo interface is used with the Apple device, the calling application can and should perform custom identification and authentication. During identification the application must specify, amongst other details, how it intends to use the device, e.g. which iAP-1 messages or iAP-2 messages it will send and receive.

The [ICinemolAP::Open\(\)](#) method provides an option to perform custom identification and default authentication. Turn on the CINEMO_IAP_OPEN_IDENTIFY bit in the flags parameter and pass via the pconfig structure (in the pidentify field) an [ICinemoMetapool](#) populated with values for IAP identification. Then [ICinemolAP::Open\(\)](#) will perform the identification with the data from the meta pool and will return with [CinemoNoError](#) if identification succeeded or with some error code if not.

A full example of this identification method can be seen in the SampleIAP.cpp source code delivered with the Cinemo Media Engine SDK. In the CinemoSampleSupport library, which is also part of the Cinemo Media Engine SDK, a class for usage specific setup of the configuration metapool is included. Cinemo recommends its usage also in the customers production code. Please refer to the IAPIdentificationConf.h and IAPIdentificationConf.cpp file in the CinemoSampleSupport folder of the SDK.

5.1.3 Callback interface usage

With the call to [ICinemolAP::Open\(\)](#) a set of interface instances can be passed to the Cinemo Media Engine. All these interfaces have no implementation in the Cinemo Media Engine SDK. It is expected that the user of the SDK provides an implementation of such an interface. By providing an instance of the interface implementation a callback mechanism is established between the users code and the Cinemo Media Engine SDK. Currently the following interface definitions are provided:

1. [ICinemolAPEAPAnnounce](#) for EAP notifications.
2. [ICinemolAPLocationInfo](#) for location information notifications.
3. [ICinemolAPPowerUpdates](#) for power update notifications.
4. [ICinemolAPCommunications](#) for communications updates and list notifications.
5. [ICinemolAPVehicleStatus](#) for vehicle status update notifications.

6. [ICinemoAPBluetoothUpdates](#) for Bluetooth connection status notifications.
7. [ICinemoAPWiFiInfoSharing](#) for WiFi information sharing between accessory and device.
8. [ICinemoAPAssistiveTouch](#) for assistive touch information updates.
9. [ICinemoAPOOBTPairing](#) for out of band bluetooth pairing information.
10. [ICinemoAPVoiceOver](#) for voice over updates.
11. [ICinemoAPRouteGuidance](#) for route guidance updates.

These interfaces will be used to notify the user of Cinemo Media Engine about certain asynchronous operations that an Apple device can initiate.

With exception to the External Accessory Protocol (EAP) at any point in time only *one* callback interface instance can be present for a given Apple device connection. A subsequent call to [ICinemoAP::Open\(\)](#) will fail (and return *CinemoErrorState*) if a callback interface instance is passed again with the subsequent call.

Because the exact point in time when the Apple device will trigger such an operation is not known for some of the callback interfaces, the safest approach is to register all of these callback interfaces with the first call to [ICinemoAP::Open\(\)](#).

Unfortunately this is also a limitation from a software design point of view, because a centralized instance needs to handle all kind of different and unrelated tasks.

Because of this the [ICinemoAP](#) API provides support to register callback interfaces at subsequent calls to additional instances of the [ICinemoAP](#) interface. For the [ICinemoAPLocationInfo](#) and [ICinemoAPVehicleStatus](#) interfaces, where the start of such notifications can not be requested from the Apple device, any notification coming from the Apple device is cached in Cinemo Media Engine. Upon attachment of a corresponding callback interface instance and the presence of cached information the callback interface will be triggered once accordingly. Afterwards events will only be triggered when the Apple device sends a corresponding message.

Cinemo recommends to plan in advance the components where the functionalities provided by the callback interfaces shall be handled. If different processes shall be involved, then the solution with attaching callback interfaces with subsequent instances of [ICinemoAP](#) (connected together via Cinemos DDP protocol) is the best approach. Care must be taken to not attach the same kind of callback interface more than once for the interfaces supporting only one instance per Apple device connection. The callback interface is cleared, when the corresponding instance of [ICinemoAP](#) is closed. An other callback instance can be attached then later when opening again.

The following table shows which iAP specific callback interfaces are affected by the specialities explained above:

Interface	max no. instances	unpredictive calls	caching supported
ICinemoAPEAPAnnounce	unlimited	✓	see usage of EAP
ICinemoAPLocationInfo	1	✓	✓
ICinemoAPPowerUpdates	1		

Interface	max no. instances	unpredictive calls	caching supported
ICinemolAPCommunications	1		
ICinemolAPVehicleStatus	1	✓	✓
ICinemolAPBluetoothUpdates	1		
ICinemolAPWiFilInfoSharing	1		
ICinemolAPAssistiveTouch	1		
ICinemolAPOOBBTPairing	1		
ICinemolAPVoiceOver	1		
ICinemolAPRouteGuidance	1		

For specific usage of these callback interfaces refer to the specific description of the interface in the sections below and the [ICinemolAP::Open\(\)](#) method.

5.1.4 Support for normal external accessory protocol: EAP

When using custom identification and providing an implementation of the [ICinemolAPEAPAnnounce](#) interface it is possible to communicate with applications on the Apple device via the normal external accessory protocol (EAP). The [ICinemolAPEAPAnnounce](#) interface consists of callback functions which will be called asynchronously in order to inform the user application about the starting, stopping and data transfer in an EAP session.

Each EAP connection is identified by the session id, which is chosen by the Apple device. The details of the interfaces involved in EAP are described in dedicated sections of this document.

In order to use different external accessory protocols from different places, Cinemo supports multiple instances of [ICinemolAPEAPAnnounce](#) to be registered with the iAP connection. For this to work subsequent calls to [ICinemolAP::Open\(\)](#) on additional instances of [ICinemolAP](#) can be used. Each additional call can register its own instance of [ICinemolAPEAPAnnounce](#), but it cannot change the identification information any more. Unfortunately, there is a potential timing problem with this approach due to the way Apple devices work. If the iOS app for an identified EAP is already active on the Apple device, it will immediately initiate an EAP session. If the EAP handler is not yet known to the Cinemo Media Engine, this session initiation request will be lost. To help solving this problem the Cinemo Media Engine supports keeping EAP connections for subsequent instances of [ICinemolAPEAPAnnounce](#). Any data received on such a connection will be buffered until used later.

The Cinemo recommendation is to register an instance of [ICinemolAPEAPAnnounce](#), which handles all identified EA protocols with the first call to [ICinemolAP::Open\(\)](#). If this is not possible, then at least a proxy-instance of [ICinemolAPEAPAnnounce](#) should be used in the first call to [ICinemolAP::Open\(\)](#) which also does the identification. This instance of [ICinemolAPEAPAnnounce](#) should make all incoming EAP requests pending and immediately trigger the thread or process which will handle the protocol in the right way.

On a subsequent instance of [ICinemolAPEAPAnnounce](#), which is registered with an additional instance of [ICinemolAP](#), the still pending sessions will be announced again. Thus the initial proxy instance can start the real

protocol handler, which itself opens his own [ICinemoIAP](#) with [ICinemoAPEPAnnounce](#). The pending session will be announced again and can then be accepted by the correct handler.

5.1.5 Browsing iAP-2 devices and MediaLibrary information flags

Cinemo supports browsing of iAP-2 devices if an [ICinemoAPMediaLibraryAccess](#) interface is passed to the [ICinemoAP::StartMediaLibraryInformation\(\)](#) method. Cinemo Media Management will automatically call [ICinemoAP::StartMediaLibraryInformation\(\)](#) and take care to add such an instance of [ICinemoAPMediaLibraryAccess](#). All instances of [ICinemoPlaylist](#) or [ICinemoVFS](#) opening this iAP device will be presented with a device browsing hierarchy (as known from iAP-1 devices). Since Cinemo Media Management is used, the browse hierarchy will be available as soon as the iAP-2 device has been mounted using the [ICinemoMM::AddIndexedVolume\(\)](#) method.

Apple devices may contain several “remote” items, which need an active internet connection for playback, e.g. “iTunes Radio” or “cloud” items. While Cinemo will automatically hide unavailable items in the device browser, they will still be available when browsing the MM server.

Cinemo will expose information about the current state of the Apple device when directly browsing the device or when using a media management server.

Directly browsing the Apple device

When browsing the root folder of an Apple device, there will be a *VFS_IAP_FLAGS* entry for each media library available on the device. These flags do contain a bitmask of *CINEMO_VFS_IAP_FLAGS* as defined in *cinemo_iap.h*.

```
# SampleMetadata "iap://usb:///dev/bus/usb/001/008"

path: iap://usb:///dev/bus/usb/001/008
type: FOLDER | VIRTUAL | ORDERED | WATCHABLE | SELECTABLE | USB | IAP | iAP-2

title 00 index 00 VFS_UUID: "369de550668861746d81c1abb6e9d212f22693f6_iAP2"
title 00 index 00 VFS_NAME: "Cinemo's iPod touch"
title 00 index 00 VFS_LANGUAGE: "en"
title 00 index 00 VFS_IAP_FLAGS: 0
title 00 index 00 NowPlayingAppDisplayName: "Music"
title 00 index 00 NowPlayingAppBundleName: "com.apple.Music"
title 01 index 00 VFS_PATH: "iaptrack://usb:///dev/bus/usb/001/008"
title 01 index 00 VFS_NAME: "NowPlaying"
title 01 index 00 VFS_TYPE: FILE | VIRTUAL | PLAYABLE | SELECTABLE | USB | IAP | iAP-2
title 01 index 00 VFS_NOWPLAYING_TYPE: 2
title 02 index 00 VFS_PATH: "iap://usb:///dev/bus/usb/001/008?pid=0"
title 02 index 00 VFS_NAME: "NowPlaying"
title 02 index 00 VFS_TYPE: FOLDER | VIRTUAL | ORDERED | WATCHABLE | SELECTABLE | USB | IAP | iAP-2
title 02 index 00 VFS_NOWPLAYING_TYPE: 1
title 03 index 00 VFS_PATH: "iap://usb:///dev/bus/usb/001/008?pid=1"
title 03 index 00 VFS_NAME: "Cinemo's iPod touch"
title 03 index 00 VFS_TYPE: FOLDER | VIRTUAL | ORDERED | WATCHABLE | SELECTABLE | USB | IAP | iAP-2
```

...

In this example the flags for the libraries of types 0 (Local device library) and 2 (iTunes Radio Library) are both 0. If the `MediaLibraryIsHidingRemoteItems` flag is set by the Apple device for a library, the value of the `VFS_IAP_FLAGS` will be 65536.

If this flag is set, no remote items are available for playback on the Apple device. Tracks will also contain `VFS_IAP_FLAGS`, which indicate whether or not the track is local or remote. This corresponds to the `MediaItemPropertyIsResidentOnDevice` flag from Apple.

Browsing via Cinemo MM server

When browsing an iAP-2 device via a Cinemo MM server, the `MediaLibraryIsHidingRemoteItems` flag is available on the basic container for each library, where also the media library information is stored.

```
# SampleMetadata "upnp://127.0.0.1:39766/0/MediaServer/DeviceDesc.xml?pid=1031&browse=BrowseDirectChildren"  
opening...  
path: upnp://127.0.0.1:39766/0/MediaServer/DeviceDesc.xml?pid=1031&browse=BrowseDirectChildren  
type: FOLDER | VIRTUAL | ORDERED | METADATA_COMPLETE | WATCHABLE | UPNP  
...  
title 01 index 00 VFS_PATH: "upnp://127.0.0.1:39766/0/MediaServer/DeviceDesc.xml?pid=1032&browse=BrowseDirectChildren"  
title 01 index 00 VFS_NAME: "Cinemo's iPod touch"  
title 01 index 00 VFS_TYPE: FOLDER | VIRTUAL | ORDERED | METADATA_COMPLETE | WATCHABLE | UPNP  
title 01 index 00 VFS_IAP2_LIBRARY_ID: "CB4905C0-83E5-4DA7-B83E-CE9FB31A851B-8.1.3"  
title 01 index 00 VFS_IAP2_LIBRARY_TYPE: 0  
title 01 index 00 VFS_IAP2_LIBRARY_REVISION: "33056"  
title 01 index 00 VFS_IAP_FLAGS: 0  
...  
title 02 index 00 VFS_PATH: "upnp://127.0.0.1:39766/0/MediaServer/DeviceDesc.xml?pid=1033&browse=BrowseDirectChildren"  
title 02 index 00 VFS_NAME: "iTunes Radio"  
title 02 index 00 VFS_TYPE: FOLDER | VIRTUAL | ORDERED | METADATA_COMPLETE | WATCHABLE | UPNP  
title 02 index 00 VFS_IAP2_LIBRARY_ID: "CB4905C0-83E5-4DA7-B83E-CE9FB31A851B-4954524C-8.1.3"  
title 02 index 00 VFS_IAP2_LIBRARY_TYPE: 2  
title 02 index 00 VFS_IAP2_LIBRARY_REVISION: "2"  
title 02 index 00 VFS_IAP_FLAGS: 0  
...
```

5.2 ICinemoAppleAuth

The [ICinemoAppleAuth](#) interface provides serialized access to the Apple authentication chip through the Cinemo API.

ICinemoAppleAuth Methods

METHOD	DESCRIPTION
<u>Open</u>	Open an Apple authentication chip interface
<u>GetSerialNumber</u>	Get the authentication chip serial number
<u>GetCertificate</u>	Get the authentication chip X.509 certificate
<u>GetChallengeResponse</u>	Get the authentication challenge response

5.2.1 Open

Open an Apple authentication chip interface.

Syntax

```
CinemoError Open(  
    [in] const char * szurl,  
    [in] const char * szparam,  
    [in] uint32 flags  
) ;
```

Parameters

- **szurl**

The URL of the Apple authentication chip.

- **szparam**

Additional parameters.

- **flags**

Bitmask of additional open flags.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The *flags* parameter supports the following bits:

```
typedef enum {  
    CINEMO_APPLE_AUTH_OPEN_DEFAULT,  
    CINEMO_APPLE_AUTH_OPEN_DEVICE  
} CINEMO_APPLE_AUTH_OPEN;
```

If *CINEMO_APPLE_AUTH_OPEN_DEVICE* is set, this instance of [ICinemoAppleAuth](#) will act as the lifecycle instance. As soon as this instance is closed, the access to the chip will be terminated.

5.2.2 GetSerialNumber

Get the authentication chip serial number.

Syntax

```
CinemoError GetSerialNumber(  
    [in] void * pbits,  
    [in] uint32 nsize,  
    [out] uint32 * pread  
) ;
```

Parameters

- **pbits**

Buffer for the serial number.

- **nsize**

Buffer size in bytes.

- **pread**

Number of bytes read into buffer.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

5.2.3 GetCertificate

Get the X.509 certificate from the Apple Authentication Coprocessor.

Syntax

```
CinemoError GetCertificate(  
    [in] void * pbits,  
    [in] uint32 nsize,  
    [out] uint32 * pread  
) ;
```

Parameters

- **pbits**

Buffer for the certificate.

- **nsize**

Buffer size in bytes.

- **pread**

Number of bytes read into buffer.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

5.2.4 GetChallengeResponse

Get the authentication challenge response.

Syntax

```
CinemoError GetChallengeResponse (
    [in] void * pbits,
    [in] uint32 nsize,
    [out] uint32 * pread,
    [in] const void * pchallenge,
    [in] uint32 nbytes
);
```

Parameters

- **pbits**

Buffer for the certificate.

- **nsize**

Buffer size in bytes.

- **pread**

Number of bytes read into buffer.

- **pchallenge**

Authentication challenge data.

- **nbytes**

Authentication challenge data size in bytes.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

5.3 ICinemolAP

The [ICinemolAP](#) interface provides operations related only to Apple devices. The Apple device specific functions take away the need to know the exact methods and commands of Apple's MFi interface and firmware specifications, while allowing specialized functionality.

The initial [ICinemolAP](#) instance must be used to control the duration of the life cycle of the connection to the Apple device maintained internally in Cinemo Media Engine.

The [ICinemolAP](#) interface should be used for connecting to Apple devices in almost every serious use case. It provides the means to configure the Apple device connection according the desired usage. Apple certification prohibits configuring a connection for certain functionality and not using it later. So Cinemo is not able to provide a default and tested working setup internally.

Technically it is not necessary to use [ICinemolAP](#) for multimedia only functions such as browsing, playback and indexing of media contents, since these functions are already fully implemented by standard Cinemo interfaces [ICinemoPlaylist](#), [ICinemoPlayer2](#) and [ICinemoMM](#). But even in this case some of the possible functions may not be used via the Cinemo Media Engine API and should be excluded in the configuration for the connection to the Apple device. This can only be achieved by using an initial instance of [ICinemolAP](#) for connection setup.

The [ICinemolAP](#) interface is instantiated by calling [CinemoCreateIAP\(\)](#) function.

ICinemolAP Methods

METHOD	DESCRIPTION
Open	Open an Apple device
Close	Close the Apple device handle
IdentificationInformationUpdate	Send an IdentificationInformationUpdate.
StartHID	Start HID messages via iAP2 protocol layer.
SendHIDReport	Send a HID report for one of the HID components configured in identification settings.
StopHID	Stop HID messages via iAP2 protocol layer.
ResumeDefaultPlayback	Switch the Apple device into default playing mode
RequestAppLaunch	Request from the Apple device to launch a certain iOS application
EAPSessionStatus	Define the status of an EAP session
EAPSessionRecv	Receive packets from the EAP app on the Apple device
EAPSessionPost	Post data packets to the EAP app on the Apple device
EAPSessionSend	Send data packets to the EAP app on the Apple device
SendLocationInformation	Send LocationInformation data messages.
StartMediaLibraryInformation	Start receiving information about the media libraries on the Apple device
StopMediaLibraryInformation	Stop receiving media library information from the Apple device

METHOD	DESCRIPTION
<u>StartMediaLibraryUpdates</u>	Start receiving information about the media items on the Apple device
<u>StopMediaLibraryUpdates</u>	Stop receiving media item information from the Apple device
<u>SetMediaLibraryStatus</u>	Set the status of the Media Library Access interface
<u>SignalMediaLibraryQueryUpdate</u>	Signal that an update changed the contents of a query to the Media Library
<u>SetMediaLibraryProgress</u>	Set the indexing progress of the Media Library
<u>SendPlayMediaLibraryCommand</u>	Send special play commands using Media Library information
<u>StartPowerUpdates</u>	Start power update messages.
<u>StopPowerUpdates</u>	Stop power update messages.
<u>SendPowerSourceUpdate</u>	Send a power source update message.
<u>StartCallStateUpdates</u>	Start phone call state updates from the Apple device.
<u>StopCallStateUpdates</u>	Stop phone call state updates from the Apple device.
<u>StartCommunicationsUpdates</u>	Start communications status update messages.
<u>StopCommunicationsUpdates</u>	Stop communications status update messages.
<u>StartListUpdates</u>	Start communications list update messages.
<u>StopListUpdates</u>	Stop communications list update messages.
<u>InitiateCall</u>	Initiate a phone call via the connected Apple device.
<u>AcceptCall</u>	Accept an incoming call.
<u>EndCall</u>	End a phone call on the connected Apple device.
<u>SwapCalls</u>	Swap phone calls on a connected Apple device.
<u>MergeCalls</u>	Merge phone calls on a connected Apple device.
<u>HoldStatusUpdate</u>	Update the hold status of a call.
<u>MuteStatusUpdate</u>	Update the mute status of a call.
<u>SendDTMF</u>	Send DTMF tone through existing call on the Apple device.
<u>SendVehicleStatus</u>	Send Vehicle Status messages.
<u>BluetoothComponentInformation</u>	Send initial Bluetooth component status.
<u>StartBluetoothConnectionUpdates</u>	Request from Apple device Bluetooth updates.
<u>StopBluetoothConnectionUpdates</u>	Stop Bluetooth updates.
<u>RequestWiFiInformation</u>	Request WiFi information from the Apple Device.
<u>AccessoryWiFiConfigurationInformation</u>	Report the WiFi configuration to the Apple Device.
<u>StartAssistiveTouch</u>	Enables assistive touch feature of the Apple Device.
<u>StopAssistiveTouch</u>	Disables assistive touch feature of the Apple Device.
<u>StartAssistiveTouchInformation</u>	Enables assistive touch information from the Apple Device.
<u>StopAssistiveTouchInformation</u>	Disables assistive touch information from the Apple Device.

METHOD	DESCRIPTION
<u>OOBBTParingAccessoryInformation</u>	Transfer bluetooth accessory information to the Apple device.
<u>OOBBTPairingCompletionInformation</u>	Report result of storing pairing link key information.
<u>StartRouteGuidanceUpdates</u>	Start Route Guidance feature on the Apple device.
<u>StopRouteGuidanceUpdates</u>	Stop Route Guidance feature on the Apple device.

5.3.1 Open

Open an Apple device for operation.

Syntax

```
CinemoError Open (
    [in] const char * szurl,
    [in] uint32 flags,
    [in] CinemoIAPConfig *pconfig,
    [out] CinemoIAPAttributes& attrs
);
```

Parameters

- **szurl**

The URL of the Apple device.

- **flags**

Bitmask of CINEMO_IAP_FLAGS for opening the device. Defaults to CINEMO_IAP_OPEN_DEFAULT.

- **pconfig**

Pointer to structure filled with configuration information. Can be NULL. The access to this structure and its individual fields depends on the given flags.

- **attrs**

Return attributes of the device.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The URL of the device needs to have either the "iap://" or "iaptrack://" protocol, otherwise this method will return *CinemoErrorArguments*.

The flags parameter determines different operation modes of the [Open\(\)](#) method:

CINEMO_IAP_OPEN_AUTHENTICATE perform Apple device authentication, if not yet done by a call to [Open\(\)](#) on an other instance of [ICinemolAP](#), [ICinemoVFS](#) or [ICinemoPlaylist](#).

CINEMO_IAP_OPEN_IDENTIFY perform identification, if not yet done by a call to [Open\(\)](#) on an other instance of [ICinemolAP](#), [ICinemoVFS](#) or [ICinemoPlaylist](#).

CINEMO_IAP_OPEN_DEVICE make this instance of [ICinemolAP](#) the iAP connection life cycle controlling instance.

CINEMO_IAP_OPEN_FOR_CARPLAY use a variant of iAP connection suitable for parallel use of CarPlay. Will have an effect only if this is also the identifying instance of the iAP connection.

CINEMO_IAP_OPEN_FINISH_AUTH iAP-1 authentication is a background process. With this flag set, the [Open\(\)](#) call will return after this process is complete, otherwise already when the minimum necessary steps are complete.

CINEMO_IAP_OPEN_REOPEN reopen the same instance if [ICinemoIAP](#) after a previously call to [Close\(\)](#) and assume the underlying transport to the Apple device (e.g. USB or Bluetooth) was not interrupted.

CINEMO_IAP_OPEN_DEFAULT default functionality performed by [Open\(\)](#) and is a combination of flags from the bits described above.

The following previously existing flag has been deprecated, but is still available for backward compatibility. You should change your code to not use this flag any more. See [section 5.3.1](#) for a more flexible replacement of this functionality.

CINEMO_IAP_OPEN_DISABLE_MEDIA_ENGINE (deprecated) prevent Cinemo from using any functions needed for media playback on this iAP connection. Using such media engine functionality with the Cinemo API will result in Cinemo error codes.

When the flags parameter is *CINEMO_IAP_OPEN_AUTHENTICATION* (which is currently also the same value as *CINEMO_IAP_OPEN_DEFAULT*) then the pconfig parameter is not evaluated and can be NULL. If the flags parameter has specific bits set, then the pconfig parameter can be used to pass optional information to the Open method.

The CinemoIAPConfig structure is currently defined as follows:

```
typedef struct {
    const char * transport_options;
    ICinemoMetapool * identify;
    ICinemoIAPEAPAnnounce * eap;
    ICinemoIAPLocationInfo * loc;
    ICinemoIAPPowerUpdate * power;
    ICinemoIAPCommunications * communications;
    ICinemoIAPVehicleStatus * vehicle;
    ICinemoIAPBluetoothUpdates * bluetooth;
    ICinemoIAPWiFiInfoSharing * wifi;
    ICinemoIAPAssistiveTouch * assistive_touch;
    ICinemoIAPOOBTPairing * oob_bt_pairing;
    ICinemoIAPVoiceOver * voice_over;
    ICinemoIAPRouteGuidance * routeguidance;
} CinemoIAPConfig;
```

It can hold pointers to different interfaces as well as transport specific options for this iAP device. The transport options are described in detail in the Transport Library documentation.

The interfaces are necessary for enabling certain iAP features, which are briefly described in the following paragraphs. These features are explained in separate subsections, but are used – when needed – finally together in one single call to [Open\(\)](#).

Control of iAP connection life cycle with Open

If the flag `CINEMO_IAP_OPEN_DEVICE` is passed via the flags argument in the call to [Open\(\)](#), then after all other instances of Cinemo API using the same Apple device have been closed, a closing of this instance will also terminate the communication with the Apple device. Without this flag, the underlying connection is kept alive even if no Cinemo API instance is referring to it. That way no additional identification needs to be performed if a new Cinemo API instance starts using the Apple device again.

The `CINEMO_IAP_OPEN_DEVICE` flag can be passed to the [Open\(\)](#) call on a subsequent created instance of [ICinemoIAP](#). That instance will become the life cycle handle of the Apple device connection and should be closed after all other Cinemo API instances referring to the Apple device have been closed.

Setup of identification in Open

If the flags parameter passed to the [Open\(\)](#) has the `CINEMO_IAP_OPEN_IDENTIFY` bit set, then either Cinemo internal or user customized accessory identification is performed. If pconfig is NULL or the pidentify field is NULL, then the Cinemo internal identification is used. If pconfig is not NULL and the pidentify field points to an [ICinemoMetapool](#) instance filled with configuration values for the iAP identification process, then the identification process is performed with the settings from the meta pool instance. More details on how to use this flag and the pidentify field can be obtained from the SampleIAP.cpp example code distributed with the Cinemo Media Engine SDK.

Detecting if the same device is opened a second time

Assume the following situation: while a connection to an Apple device is open (e.g. via Bluetooth) a new device appears on the USB bus because the user plugged the same Apple device also into the USB-Connector. In this case it is important to determine if the same Apple Device (which is already used for media playback over the other transport protocol (I.E. Bluetooth)) has been attached.

The Apple devices supporting only the iAP1 protocol have the limitation that not all features of iAP1 can be identified for the same device a second time. Thus handling this situation is dependent on the version of the iAP protocol supported by the Apple device. The protocol version can only be detected during the call to [Open\(\)](#), which makes it difficult to decide how to configure the identification settings. Cinemo has built in support to handle this situation. You must use a customized identification pool, which has to be passed in the pidentify field of the pconfig parameter of the [Open\(\)](#) method.

Please follow the procedure outlined below. For every step it is given if it applies to iAP1, iAP2 or both.

1. <iAP1,iAP2> Add all your needed entries which don't depend on the version of the iAP protocol into an initial empty instance of [ICinemoMetapool](#) named "pool" in the following sections.
2. <iAP1,iAP2> If you intend to use the EAP feature, then add the corresponding configuration items into the pool.
3. <iAP1> Add a minimal set of entries into the pool needed for iAP1 the pool.

4. <iAP1> add an integer with the meta name `CINEMO_METANAME_IAP_IAP1_CONFIG_FLAGS` and a value which contains the following bits:

`CINEMO_IAP_CONFIG_RESTRICTED`: this connection shall be used for device identification.

`CINEMO_IAP_CONFIG_POWER_CHARGE`: If this connection is via USB (e.g. wired) then also apply the power charge configuration settings.
5. <iAP2> Add a full featured set of identification setting for iAP2 into the pool.
6. <iAP1,iAP2> Call [Open\(\)](#) with the configured pool and an open flag where the `CINEMO_IAP_OPEN_IDENTIFY` bit is set. Since this will also be the first instance of [ICinemoIAP](#) to communicate with the Apple device, you should use this also to control the lifecycle of the connection by adding the `CINEMO_IAP_OPEN_DEVICE_LIFECYCLE` flag.
7. <iAP1,iAP2> If [Open\(\)](#) returns successful the resulting attributes parameter indicates which version of iAP is now used with the connected Apple device.

At this point the Apple device is connected and identified either with all needed features (iAP2) or only in a restricted mode (iAP1). You also know which kind of connection it is when looking at the value of the attributes parameter.

The restricted identification for iAP1 Apple devices is not sufficient to be used with Cinemo Media Engine functionality. The error code `CinemoErrorState` is returned if functions on instances of [ICinemoPlaylist](#) or [ICinemoPlayer2](#) or other Cinemo functionality which need the Cinemo Media Engine setup are called, while the Apple Device is only identified with a restricted configuration set.

Now create an instance of [ICinemoVFS](#). Call on this instance [ICinemoVFS::Open\(\)](#) with the same URI as used for the call to [Open\(\)](#) and retrieve its meta pool. Extract from this device meta pool the entry `CINEMO_METANAME_DEVICE_IDENTIFICATION`. Use the value of this entry to compare against the equivalent value of the already existing connection to the Apple device. If the values are equal, then the same device was connected through two different transports (e.g. USB and Bluetooth). Don't use this value for further device identification, as it is not persistent (even for the same device) after the device has been disconnected.

Apple iAP2 specification states that it is not allowed to have more than one transport to be active for an extended period, and that wired connections must be preferred. At this point a decision must be made whether the new connection shall be used from now on, or the old shall be kept.

But it is possible to use the restricted iAP1 connection to power charge the Apple device. Other functionality is not usable. An example of this use case is available in the SampleIAP.cpp and command_power.cpp example source codes provided with the Cinemo SDK.

If the new connection to the Apple device shall not be used, then call [Close\(\)](#) and release the instance of [ICinemoIAP](#).

If the switch to the new connection shall be applied, then call [Close\(\)](#) on the old, previously existing instance of [ICinemoIAP](#) and release it.

In case of iAP2 the newly opened instance of [ICinemoIAP](#) is full featured and can be used. In case of iAP1 it needs to be upgraded to full feature with the following procedure:

1. <iAP1> Add all your needed entries which don't depend on the version of the iAP protocol into an initial empty instance of [ICinemoMetapool](#) named "pool" in the following sections.
2. <iAP1> If you intend to use the EAP feature, then add the corresponding configuration items into the pool.
3. <iAP1> Add a full featured set of identification setting for iAP1 into the pool. This feature set may also contain generic setting for the EAP feature.
4. <iAP1> Call [Open\(flags=\(CINEMO_IAP_OPEN_IDENTIFY|CINEMO_IAP_OPEN_REOPEN\)\)](#) with the configured pool and open flags where the following bits are set:
 - CINEMO_IAP_OPEN_IDENTIFY:** perform identification.
 - CINEMO_IAP_OPEN_REOPEN:** indicate that device is already open.
5. <iAP1> If [Open\(with CINEMO_IAP_OPEN_REOPEN flag\)](#) returns successful the Apple device connection has been upgraded to support the features needed for the Cinemo Media Engine functionality.

At this point the new [ICinemoIAP](#) instance – provided all functions returned *CinemoNoError* – is open for usage with the connected Apple device regardless of the used iAP version (either iAP1 or iAP2).

Setup of EAP with Open

In order to use the Cinemo built-in EAP support the following steps need to be followed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoAPEPAnnounce](#) interface.
2. Pass one instance of the implementation of the [ICinemoAPEPAnnounce](#) interface via the eap field of the CinemoIAPConfig structure at the call to Open.
3. Declare the correct EAP settings in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed, then whenever activity regarding one EAP sessions occurs, the corresponding function from [ICinemoAPEPAnnounce](#) will be called asynchronously by the Cinemo Media Engine. It is possible that the callback functions from the [ICinemoAPEPAnnounce](#) interface are triggered before the ongoing call to [Open\(\)](#) returns. This is a valid use case supported by Cinemo. The functions of the of [ICinemoIAP](#) instance needed as response from the callback are already available at this time. Please note that sending data on a retrieved EA session can only start after the [ICinemoAPEPAnnounce::EAPSessionInitiated\(\)](#) returned successful.

Setup of Location Information with Open

In order to use the Cinemo built-in location support the following steps need to be followed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoAPLocationInfo](#) interface.
2. Pass one instance of the implementation of the [ICinemoAPLocationInfo](#) interface via the loc field of the CinemoAPConfig structure at the call to Open.
3. Declare the correct location information settings in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed, Cinemo Media Engine will use the [ICinemoAPLocationInfo](#) interface to inform asynchronously about location information sessions started or stopped by the Apple device. It is possible that the callback functions from the [ICinemoAPLocationInfo](#) interface are triggered before the ongoing call to [Open\(\)](#) returns. This is a valid use case supported by Cinemo. The functions of the of [ICinemoAP](#) instance needed as response from the callback are already available at this time.

Setup of Power updates with Open

Power updates will be available if the following steps are followed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoAPPowerUpdate](#) interface.
2. Pass one instance of the implementation of the [ICinemoAPPowerUpdate](#) interface via the power field of the CinemoAPConfig structure at the call to Open.
3. Declare the correct power update settings in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed and the call to [Open\(\)](#) returns successful, [StartPowerUpdates\(\)](#) and [StopPowerUpdates\(\)](#) are available to start power sessions on the Apple device. Cinemo Media Engine will use the [ICinemoAPPowerUpdate](#) interface to inform asynchronously about power messages sent by the Apple device. Additionally the [SendPowerSourceUpdate\(\)](#) method can be used to send power information to the Apple device. This particular function is already available even before the call to [Open\(\)](#) returns successful, because of Apple device timing requirements.

Setup of Communications updates with Open

In order to use the Cinemo built-in iAP communications support the following steps need to be followed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoAPCommunications](#) interface.
2. Pass one instance of the implementation of the [ICinemoAPCommunications](#) interface via the com field of the CinemoAPConfig structure at the call to Open.

3. Declare the correct communications settings in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed and the call to [Open\(\)](#) returns successful, [StartCallStateUpdates\(\)](#), [StartCommunicationsUpdates\(\)](#) and [StartCommunicationsListUpdates\(\)](#) are available to start communications information on the Apple device. Cinemo Media Engine will use the [ICinemoIAPCommunications](#) interface to inform asynchronously about communications information and status updates from the device.

Setup of Vehicle Status with Open

In order to use the Cinemo built-in vehicle status support the following steps need to be followed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoIAPVehicleStatus](#) interface.
2. Pass one instance of the implementation of the [ICinemoIAPVehicleStatus](#) interface via the vehicle field of the CinemoIAPConfig structure at the call to Open.
3. Declare the correct vehicle status settings in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed, Cinemo Media Engine will use the [ICinemoIAPVehicleStatus](#) interface to inform asynchronously about vehicle status sessions started or stopped by the Apple device. It is possible that the callback functions from the [ICinemoIAPVehicleStatus](#) interface are triggered before the ongoing call to [Open\(\)](#) returns. This is a valid use case supported by Cinemo. The functions of the of [ICinemoIAP](#) instance needed as response from the callback are already available at this time.

Bluetooth connection updates

In order to use the Cinemo built-in Bluetooth connection updates support the following steps need to be followed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoIAPBluetoothUpdates](#) interface.
2. Pass one instance of the implementation of the [ICinemoIAPBluetoothUpdates](#) interface via the bluetooth field of the CinemoIAPConfig structure at the call to Open.
3. Declare the correct bluetooth components in the configuration meta pool passed for accessory identification (see above) during the call to open.
4. Immediately after the call to Open an initial call to [BluetoothComponentInformation\(\)](#) must declare the initial local view of the Bluetooth components enumerated in the identification customization meta pool.

When these steps have been followed, Cinemo Media Engine will use the [ICinemoIAPBluetoothUpdates::BluetoothComponentStatus\(\)](#) call to inform asynchronously about bluetooth connection updates initiated by the Apple device. For iAP1 some tight timing

requirements apply for sending the initial [BluetoothComponentInformation\(\)](#). Therefore the [ICinemoIAPBluetoothUpdates::InitialBluetoothComponentInformationRequest\(\)](#) callback might be called during an ongoing [Open\(\)](#) call, which indicates the need to immediately send the desired initial set of bluetooth component information settings with a call to [BluetoothComponentInformation\(\)](#).

Setup of WiFi information sharing with Open

In order to use the Cinemo built in WiFi information sharing support the following steps need to be followed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoIAPWiFilInfoSharing](#) interface.
2. Pass one instance of the implementation of the [ICinemoIAPWiFilInfoSharing](#) interface via the wifi field of the CinemoIAPConfig structure at the call to Open.
3. Declare the correct WiFi settings in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed, Cinemo Media Engine will use the [ICinemoIAPWiFilInfoSharing](#) interface to inform asynchronously about WiFi information from the Apple device. It is possible that the callback functions from the [ICinemoIAPWiFilInfoSharing](#) interface are triggered before the ongoing call to [Open\(\)](#) returns. This is a valid use case supported by Cinemo. The functions of the of [ICinemoIAP](#) instance needed as response from the callback are already available at this time.

Setup of Assistive Touch with Open

Assistive Touch messages will be available if the following steps are performed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoIAPAssistiveTouch](#) interface.
2. Pass one instance of the implementation of the [ICinemoIAPAssistiveTouch](#) interface via the assistive_touch field of the CinemoIAPConfig structure at the call to Open.
3. Declare the correct assistive touch messages in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed and the call to [Open\(\)](#) returns successful the assistive touch related messages are available to use assistive touch with the Apple device. Cinemo Media Engine will use the [ICinemoIAPAssistiveTouch](#) interface to inform asynchronously about assistive touch information messages sent by the Apple device.

Setup of Out-Of-Band Bluetooth Pairing with Open

Out-of-Band Bluetooth Pairing messages will be available if the following steps are performed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoIAPOOBTPairing](#) interface.

2. Pass one instance of the implementation of the [ICinemoAPOOBTPairing](#) interface via the oob_bt_pairing field of the CinemoIAPConfig structure at the call to Open.
3. Declare the correct out-of-band bluetooth pairing messages in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed and the call to [Open\(\)](#) returns successful the out-of-band bluetooth pairing related messages are available to perform bluetooth pairing with the Apple device. Cinemo Media Engine will use the [ICinemoAPOOBTPairing](#) interface to inform asynchronously about out-of-band bluetooth pairing messages sent by the Apple device.

Setup of Route Guidance with Open

Route Guidance messages will be available if the following steps are performed when using the [Open\(\)](#) method.

1. Implement a class derived from the [ICinemoAPRouteGuidance](#) interface.
2. Pass one instance of the implementation of the [ICinemoAPRouteGuidance](#) interface via the routeguidance field of the CinemoIAPConfig structure at the call to [Open\(\)](#).
3. Declare the correct Route Guidance messages in the configuration meta pool passed for accessory identification (see above) during the call to open.

When these steps have been followed and the call to [Open\(\)](#) returns successful the Route Guidance related messages are available in the connection with the Apple device. After calling [ICinemoAP::StartRouteGuidanceUpdates\(\)](#) Cinemo Media Engine will use the [ICinemoAPRouteGuidance](#) interface to inform asynchronously about Route Guidance messages sent by the Apple device.

Retrieving information about the connected iAP device

The CinemoIAPAttributes structure is defined as follows:

```
typedef struct {
    uint32 type;
    uint8 iap2_file_transfer_session_version;
} CinemoIAPAttributes;
```

On successful return, CinemoIAPAttributes will be initialized with the current status of the Apple device. The field *type*, of type *CINEMO_VFS_TYPE*, can be used to determine if the device is using iAP-1 or iAP-2 protocol, as demonstrated by the following code snippet:

```
CinemoIAPAttributes attrs;
CinemoAutoPtr<ICinemoIAP> piap;
CinemoError hr;

if (hr = CinemoCreateIAP(piap.pp())) {
    printf("CinemoCreateIAP() returned %s\n", CinemoErrorToString(hr));
    return hr;
```

```
}

if (hr = piap->Open(
    szurl,
    CINEMO_IAP_OPEN_DEFAULT,
    NULL,
    attrs)) {
    printf("ICinemoIAP::Open() returned %s\n", CinemoErrorToString(hr));
    return hr;
}

switch (attrs.type & CINEMO_VFS_PROTOCOL_MASK) {
case CINEMO_VFS_PROTOCOL_IAP1:
    /* iAP-1 */
    break;
case CINEMO_VFS_PROTOCOL_IAP2:
    /* iAP-2 */
    break;
}
```

If the device is using iAP-1 transport protocol, then only methods supported by iAP1 may be used; if the device is iAP-2, then only methods supported by iAP2 may be used. If the device supports both protocols, then one or the other will be selected on opening the device, as specified by Cinemo configuration options. It is not possible to dynamically change the protocol at runtime.

5.3.2 Close

Close the handle to the Apple device connection and free all associated resources.

Syntax

```
CinemoError Close();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

After this method the instance of [ICinemolAP](#) must be released and no other method should be called.

5.3.3 IdentificationInformationUpdate

Send an IdentificationInformationUpdate to the device.

Syntax

```
CinemoError IdentificationInformationUpdate(
    const char * name,                  /* [in] Name */
    const char * model_identifier,      /* [in] ModelIdentifier */
    const char * manufacturer,         /* [in] Manufacturer */
    const char * serial_number,        /* [in] SerialNumber */
    const char * firmware_version,     /* [in] FirmwareVersion */
    const char * hardware_version,     /* [in] HardwareVersion */
    const char * current_language     /* [in] CurrentLanguage */
);
```

Parameters

- **name**

Accessory name. A blank string is not allowed.

- **model_identifier**

Accessory model name. A blank string is not allowed.

- **manufacturer**

Accessory manufacturer. A blank string is not allowed.

- **serial_number**

Accessory serial number. A blank string is not allowed.

- **firmware_version**

Accessory firmware version. A blank string is not allowed.

- **hardware_version**

Accessory hardware version. A blank string is not allowed.

- **current_language**

Accessory current active language. Must be one of the supported languages.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may be used to send updated Identification information to the iAP device.

Only iAP2 devices support this method, iAP1 devices will return *CinemoErrorDeviceSupportMissing*.

5.3.4 StartHID

Start HID session via iAP2 protocol layer for a specific HID component.

Syntax

```
CinemoError StartHID(  
    [in] const char * component_name           /* HID component name */  
) ;
```

Parameters

- **component_name**

The HID component identifier name.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may be used to start a HID session on the iAP2 layer for a previously registered HID component. If the device is connected via iAP1 then the return value will be *CinemoErrorDeviceSupportMissing*.

Before sending HID reports via [SendHIDReport\(\)](#) on the iAP2 protocol layer the HID session needs to be announced to the iAP device.

For HID reports on native transport layers this method will return *CinemoErrorState*, the sessions will automatically be started by the iAP device.

5.3.5 SendHIDReport

Send a HID report for one of the HID components configured in identification settings.

Syntax

```
CinemoError SendHIDReport(
    [in] const char * component_name,      /* HID component name */
    [in] const void * preport,            /* HID report bytes */
    [in] uint32 nbytes                  /* HID report size */
);
```

Parameters

- **component_name**

The HID component identifier name.

- **preport**

The actual HID report.

- **nbytes**

The HID report size.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This function can be used to send arbitrary HID reports to the connected iAP2 device. If the device is connected via iAP1 then the return value will be *CinemoErrorDeviceSupportMissing*.

Prior to using this function corresponding HID components need to be registered via the identification metapool provided to [Open\(\)](#). If no customized identification was used, then no HID reports may be sent and the function returns with the error *CinemoErrorState*.

In the case that a “playback remote HID descriptor” is present Cinemo will generate HID reports internally when the Apple devices needs to be remote controlled as a consequence from calls to other Cinemo API methods such as in ICinemoPlaylist and ICinemoPlayer2 (or even ICinemoPlayer). Especially the [ICinemoPlayer2::PostKeyUserEvent\(\)](#) will generate Cinemo internal HID reports.

The internal reports and HID reports for the playback remote HID descriptor will be merged, as each report describes a change in button states.

5.3.6 StopHID

Stop HID session via iAP2 protocol layer for a specific HID component.

Syntax

```
CinemoError StopHID(  
    [in] const char * component_name           /* HID component name */  
) ;
```

Parameters

- **component_name**

The HID component identifier name.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method may be used to stop a HID session on the iAP2 layer for a previously started HID component. If the device is connected via iAP1 then the return value will be *CinemoErrorDeviceSupportMissing*.

For HID reports on native transport layers this method will return *CinemoErrorState*, the sessions don't need to be stopped.

5.3.7 ResumeDefaultPlayback

Switch the apple device into default playing mode if possible.

Syntax

```
CinemoError ResumeDefaultPlayback();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will try to resume the default playback application on the Apple device. This default application is on most Apple devices the “Music” app. The operation succeeds only under certain circumstances defined by the Apple device itself. When the operation failed a *CinemoError* value is returned. But even when the operation returns with *CinemoNoError* the user of this method can only be sure that the default playback application is running by monitoring the CINEMO_EC_METADATA events on [ICinemoPlayer](#) and observing the “NowPlayingAppName” meta value for its content. On iAP-1 devices and on iAP-2 devices with iOS 8 and newer the additional “NowPlayingAppBundleName” meta value gives an even more exact definition of the currently playing application.

5.3.8 RequestAppLaunch

Send a request to the Apple device to launch a certain iOS application.

Syntax

```
CinemoError RequestAppLaunch(  
    [in] const char * appBundleId,  
    [in] CINEMO_IAP_APP_LAUNCH_ALERT alert = CINEMO_IAP_APP_LAUNCH_USER_ALERT  
) ;
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will try to launch the given iOS application on the Apple device. A successful return value of this function is **no guarantee that the application is being executed** on the apple device. On iAP1 devices the*CinemoErrorNoSuchObject* indicates that the application is missing on the device. This type of notification does not exist for iAP2 devices. On iAP1 and on iAP2 – if using the default value of the ‘alert’ parameter – a popup is displayed for the user to confirm or deny the start of the application. If the user denies the start of the application the function also returns without error. The ‘alert’ parameter is allowed to be changed from its default value only if the Apple devices has been connected via the Lightning cable.

There is no direct feedback available from Apple devices if the requested application could be started. In certain cases it may be possible to watch the content of the *NowPlayingAppName* or *NowPlayingAppBundleName* entries in the device root metapool for changes. Alternatively, if the requested application initiates an EAP session, an [*ICinemolAPEPAnnounce::EAPSessionInitiated\(\)*](#) callback would be a sure sign that the application is running.

5.3.9 EAPSessionStatus

Set the status of an announced EAP session.

Syntax

```
CinemoError EAPSessionStatus(
    [in] uint16 session_id,
    [in] CINEMO_IAP_EAP_SESSION_STATUS status
);
```

Parameters

- **session_id**

The session id number for the corresponding EAP connection.

- **status**

The desired status value (OK for accepting the session, CLOSE for denying).

Return value

The function returns *CinemoNoError* if the session status was set successfully. The function returns *CinemoErrorNoSuchObject* if the session id value is invalid. Any other return value indicates some type of error which occurred during the attempt to send the command to the Apple device.

Remarks

IMPORTANT: Do no call this function as reaction to [ICinemoIAP::EndEAPSession\(\)](#).

This function must be called on an instance of [ICinemoIAP](#) where an [ICinemoIAPAnnounce](#) instance was passed in the configuration parameters to the [Open\(\)](#) function.

This functions serves two different kind of purposes depending on the state of an EAP session.

The state of an session can be changed with the *status* parameter. The definition of *CINEMO_IAP_EAP_SESSION_STATUS* is as follows:

```
enum CINEMO_IAP_EAP_SESSION_STATUS {
    CINEMO_IAP_EAP_SESSION_STATUS_OK,
    CINEMO_IAP_EAP_SESSION_STATUS_CLOSE
};
```

To set a session into *accepted* state, the *CINEMO_IAP_EAP_SESSION_STATUS_OK* value should be used in the *status* parameter. The session is set into *denied* state with *CINEMO_IAP_EAP_SESSION_STATUS_CLOSE*.

If this function is not called, then the session will remain in *pending* state.

Every session being announced is automatically in *pending* state. If the session identified by the *session_id* parameter is in *pending* state it behaves as follows:

If *status* set the session into *accepted* state, then the associated instance of [ICinemoIAPEAPAnnounce](#) becomes the announcement target for incoming data for the session identified by the *session_id* parameter.

Only sessions in *accepted* state receive announcements of incoming data. For sessions in *pending* state Cinemo caches data sent to them, until they become *accepted*.

If set to *denied* state, the session will be ended. A corresponding call to [ICinemoIAPEAPAnnounce::EndEAPSession\(\)](#) will occur) after the Apple device acknowledged the denial of the session.

If this function is called when the EAP session is in *accepted* state in order to deny the already functional session (e.g *status* is set to *CINEMO_IAP_EAP_SESSION_STATUS_CLOSE*, then this function requests a termination of the session and the following remarks apply:

This function is not the recommended way to terminate EAP sessions. Since the EA protocol is not known to Cinemo or Apple, it remains in the responsibility of the EA session handling code to agree on a termination of the session with EA protocol specific messages exchanged through the channel provided by the EAP session. At normal EAP session termination the Apple device sends the iAP message which results in the calling of [ICinemoIAPEAPAnnounce::EndEAPSession\(\)](#) and thus allowing for all EAP session related resources to be released.

With this function the accessory side can terminate an EAP session (by force for iAP1 or with a specific iAP2 request). When calling this function a call to [ICinemoIAPEAPAnnounce::EndEAPSession\(\)](#) will occur in order to trigger the release of associated resources.

For iAP1 devices it is not possible to close an EAP session from the accessory side, thus only the resources associated with the *session_id* are removed in case of termination. For iAP2 the Apple device is being informed about the wish of the accessory side to close the EAP session. The result can be a message from the Apple device indicating the end of the EAP session, which in turn will lead to the call of the [ICinemoIAPEAPAnnounce::EndEAPSession\(\)](#). For iAP1 the call to [ICinemoIAPEAPAnnounce::EndEAPSession\(\)](#) is triggered by Cinemo internally after all session resources have been released.

For iAP2 it might be possible, that the Apple device is not informed about the termination (e.g. because of a communication error) which leads to the same situation like in the iAP1 case. In this case the EAP session still will be closed on accessory side. Any data received on a non existing or terminated EAP session will be discarded by the Cinemo Media Engine and not acknowledged towards the Apple device.

5.3.10 EAPSessionRecv

Receive packets sent from the EAP app on the Apple device.

Syntax

```
CinemoError EAPSessionRecv(
    [in]  uint16 session_id,
    [out] ICinemoBlob ** pp
);
```

Parameters

- **session_id**

The session id number for the corresponding EAP connection.

- **pp**

A pointer to a pointer of an instance of [ICinemoBlob](#). Must be non NULL.

Return value

If the *session_id* parameter does not identify a known and *accepted* EAP session then a *CinemoError* return value indicates this. Either *CinemoErrorNoSuchObject* or *CinemoErrorState* will be returned in this case. The remaining return values of this function indicates whether the returned instance of [ICinemoBlob](#) is valid or not. If case the return value is *CinemoNoError* the [ICinemoBlob](#) contains the next block of data which was not yet received by the user code.

Remarks

Cinemo recommends that this function is called on the same instance of [ICinemolAP](#) where the EAP session was set into *accepted* state.

Data coming from the EAP application on the Apple device is queued in chronological order and must be received by the user code with this function. When no data is available then this function will block until data becomes available. The function will return all received data, even if the connection was closed in the meantime.

The payload does not contain the session identifier. The iAP protocol layer information is stripped from the packet received from the Apple device before the data is passed to the user of this API function.

If the return value is *CinemoErrorCancel* then the EAP session has been closed.

A blocking call to this function will be unblocked when the EAP session is about to be closed and it will return with *CinemoErrorCancel* only after all possibly remaining non blocking calls to this function returned all pending data.

It is allowed to call this function from within the data arrival announcement of [ICinemolAPEAPAnnounce::EAPPacketArrived\(\)](#). This way an event driven, non blocking receiving of data can be implemented.

Cinemo does not recommend to mix event driven and polling receiving of EAP session data. Please consider to use the (recommended) event driven approach. If this is not feasible then using this method in polling way (it blocks until data is available) remains an option.

HUMAX Confidential

5.3.11 EAPSessionPost

Post data packets to the EAP app on the Apple device.

Syntax

```
CinemoError EAPSessionPost(
    [in] uint16 session_id,
    [in] const void * payload,
    [in] int payload_bytes
);
```

Parameters

- **session_id**

The session id number for the corresponding EAP connection.

- **payload**

A pointer to the data to be sent.

- **payload_bytes**

The number of bytes to be sent.

Return value

If the *session_id* parameter does not identify a known and *accepted* EAP session then a *CinemoError* return value indicates this. Either *CinemoErrorNoSuchObject* or *CinemoErrorState* will be returned in this case.

The function returns *Cinemo.NoError* if the data was queued to be sent to the Apple device. Any other return value indicates some type of error which occurred during the attempt to setup sending of the data to the other side of the EAP connection.

Remarks

This function cannot be called from within the execution time of the [ICinemoAPEAPAnnounce::EAPSessionInitiated\(\)](#) callback. In this case *CinemoErrorOperationProhibited* is returned.

For iAP-1 connected Apple Devices this function behaves exactly like [EAPSessionSend\(\)](#).

For iAP-2 connected Apple Devices this function behaves mostly like [EAPSessionSend\(\)](#), except the last chunk is also transferred asynchronously. This means there is no delay before this function returns for waiting the data to be acknowledged from the Apple device. But this also means: there is no guarantee that all of the data has reached the other side of the EAP connection. Possible errors that occurred while the scheduled chunks were transmitted will not be reported. Only permanent errors which prevent sending any further data will be reported with the next call to this function.

5.3.12 EAPSessionSend

Send data packets to the EAP app on the Apple device.

Syntax

```
CinemoError EAPSessionSend(
    [in] uint16 session_id,
    [in] const void * payload,
    [in] int payload_bytes
);
```

Parameters

- **session_id**

The session id number for the corresponding EAP connection.

- **payload**

A pointer to the data to be sent.

- **payload_bytes**

The number of bytes to be sent.

Return value

If the *session_id* parameter does not identify a known and *accepted* EAP session then a *CinemoError* return value indicates this. Either *CinemoErrorNoSuchObject* or *CinemoErrorState* will be returned in this case.

The function returns *CinemoNoError* if the data was sent successfully to the Apple device. Any other return value indicates some type of error which occurred during the attempt to send the data to the other side of the EAP connection.

Remarks

This function cannot be called from within the execution time of the [ICinemoIAPEAPAnnounce::EAPSessionInitiated\(\)](#) callback. In this case *CinemoErrorOperationProhibited* is returned.

If the block of data is bigger than the allowed packet size of the iAP connection the data will be split into smaller chunks and each of them will be transferred. If an error occurs during this process the function returns with the corresponding error code.

The payload *must not contain* the session identifier because this function will add the correct identifier to the chunks sent via the iAP connection to the Apple device.

For iAP-1 connected Apple devices this function sends each chunk and waits for the acknowledgement from the Apple device as required by the Apple specification. If recoverable errors are indicated, then the chunk is repeated to be sent up to the specified 10 times.

For iAP-2 connected Apple devices this function sends all but the last data chunk asynchronously to the Apple device. It then synchronously sends the last chunk to the Apple device. Through the nature of the iAP-2 link it is ensured, that the acknowledgement of the last chunk also includes the acknowledgments of the previous chunks.

HUMAX CONFIDENTIAL AUTOMOTIVE

5.3.13 SendLocationInformation

Send LocationInformation data messages to an Apple device.

Syntax

```
CinemoError SendLocationInformation(  
    [in] const int num_messages,  
    [in] const char ** szmessages  
) ;
```

Parameters

- **num_messages**

The number of messages in the szmessages array.

- **szmessages**

Array of location information messages.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

In order to provide location data to an Apple device, an instance of the ICinemoAPLocationInfo interface needs to be passed to the [Open\(\)](#) method. Once the start method of this interface has been called, [SendLocationInformation\(\)](#) needs to be called regularly with updated location information.

[SendLocationInformation\(\)](#) will return with *CinemoErrorDomain* if it is called outside of a location information session started by the Apple device.

5.3.14 StartMediaLibraryInformation

Start receiving information about the media libraries on the Apple device.

Syntax

```
CinemoError StartMediaLibraryInformation(
    [in] ICinemoIAPMediaLibraryAccess * paccess,
    [in] uint32 flags
);
```

Parameters

- **paccess**

Interface for the media library access.

- **flags**

CINEMO_IAP_LIBRARY_FLAGS bitmask.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will start MediaLibraryInformation updates from the Apple device and also register the interface for receiving these messages. The flags parameter shall be used to specify which capabilities the provided [ICinemoIAPMediaLibraryAccess](#) interface can provide.

The *CINEMO_IAP_LIBRARY_FLAGS* have the following meaning:

- **CINEMO_IAP_LIBRARY_FLAGS_ENABLE_MEDIAITEMS**

Allow the [ICinemoIAPMediaLibraryAccess::GetMediaItem\(\)](#) call. If this is not set no metadata will be available for the NowPlaying playlist. (Devices running iOS 10 or later will be able to provide a limited set of metadata for the NowPlaying playlist even without an [ICinemoIAPMediaLibraryAccess](#) interface)

- **CINEMO_IAP_LIBRARY_FLAGS_ENABLE_COLLECTION**

Allow the [ICinemoIAPMediaLibraryAccess::GetCollectionListing\(\)](#) call. If this is not set the native device browse hierarchy cannot be accessed.

Only iAP2 devices support sending MediaLibraryInformation.

5.3.15 StopMediaLibraryInformation

Stop receiving media library information from the Apple device.

Syntax

```
CinemoError StopMediaLibraryInformation();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method terminate all *MediaLibraryInformation* updates from the Apple device. The [ICinemoIAPMediaLibraryAccess](#) interface, registered with the [StartMediaLibraryInformation\(\)](#) method, will no longer be accessed after this call.

Only iAP2 devices support sending *MediaLibraryInformation*.

5.3.16 StartMediaLibraryUpdates

Start receiving information about the media items on the Apple device.

Syntax

```
CinemoError StartMediaLibraryUpdates(
    [in] const char * MediaLibraryUniqueIdentifier,
    [in] const char * LastKnownMediaLibraryRevision,
    [in] uint64 item_flags,
    [in] uint32 playlist_flags,
    [in] uint64 playlist_content_flags
);
```

Parameters

- **MediaLibraryUniqueIdentifier**

The unique identifier of the media library.

- **LastKnownMediaLibraryRevision**

The last known media library revision.

- **item_flags**

Bitmask of *CINEMO_IAP_MEDIAINFO_FLAGS*.

- **playlist_flags**

Bitmask of *CINEMO_IAP_MEDIAPLAYLIST_FLAGS*.

- **playlist_content_flags**

Bitmask of *CINEMO_IAP_MEDIAINFO_FLAGS*.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The *LastKnownMediaLibraryRevision* parameter shall be set to the last known *Revision* parameter received in a previously established iAP2 MediaLibraryUpdate session. If this parameter is *NULL*, the Apple device will begin a full database update.

The *item_flags* and *playlist_flags* parameters can be used to specify which kind of information is requested from the Apple device for the respective update messages.

Starting with iOS 10, Apple devices may include playlist content directly and not only as a list of contained UIDs. To enable this feature, the *CINEMO_IAP_MEDIAPLAYLIST_CONTAINED_ITEM_LIST* bit needs to be set in the *playlist_flags* parameter. The *playlist_content_flags* may then be used to specify which information is required. Any combination of the following bits is possible:

- *CINEMO_IAP_MEDIAINFO_PERSISTENT_IDENTIFIER*
- *CINEMO_IAP_MEDIAINFO_TITLE*

- *CINEMO_IAP_MEDIAINFO_ALBUM_TITLE*
- *CINEMO_IAP_MEDIAINFO_ARTIST*
- *CINEMO_IAP_MEDIAINFO_ALBUMARTIST*
- *CINEMO_IAP_MEDIAINFO_GENRE*
- *CINEMO_IAP_MEDIAINFO_COMPOSER*

Fields not listed here are not supported for this feature.

It is possible to specify both *CINEMO_IAP_MEDIAPLAYLIST_CONTAINED_UID_LIST* and *CINEMO_IAP_MEDIAPLAYLIST_CONTAINED_ITEM_LIST* at the same time. The Apple device will then decide how the playlist content is provided.

Only iAP2 devices support sending MediaLibraryUpdates.

5.3.17 StopMediaLibraryUpdates

Stop receiving media item information from the Apple device.

Syntax

```
CinemoError StopMediaLibraryUpdates(  
    [in] const char * MediaLibraryUniqueIdentifier  
) ;
```

Parameters

- **MediaLibraryUniqueIdentifier**

The unique identifier of the media library.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method terminate all MediaLibraryUpdates updates from the Apple device for the specified media library.

Only iAP2 devices support sending MediaLibraryUpdates.

5.3.18 SetMediaLibraryStatus

Set the status of the Media Library Access interface.

Syntax

```
CinemoError SetMediaLibraryStatus(  
    [in]  uint32 status  
) ;
```

Parameters

- **status**

Bitmask of CINEMO_IAP_MEDIALIB_STATUS.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a CinemoError code.

Remarks

This method allows to temporarily pause MediaLibraryInformation and MediaLibraryUpdates sent from the device. Please note that setting CINEMO_IAP_STOP_MEDIALIB_INFORMATION or CINEMO_IAP_STOP_MEDIALIB_UPDATES does not prevent [StartMediaLibraryInformation\(\)](#) or [StartMediaLibraryUpdates\(\)](#) from starting. Setting a stop state does also not replace a call to [StopMediaLibraryUpdates\(\)](#) or [StopMediaLibraryInformation\(\)](#), they both still are required for proper operation.

[SetMediaLibraryStatus\(\)](#) shall only be called while a device is mounted.

Only iAP2 devices support sending SetMediaLibraryStatus.

5.3.19 SignalMediaLibraryQueryUpdate

Signal that an update changed the contents of a query to the Media Library.

Syntax

```
CinemoError SignalMediaLibraryQueryUpdate(  
    [in]  uint32 queryID  
) ;
```

Parameters

- **queryID**

ID of the changed query.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method must only be used by an implementation of the [ICinemoIAPMediaLibraryAccess](#) interface. If Cinemo Media Management is used for indexing iAP devices this method will be handled internally and need not be called otherwise.

Only iAP2 devices support sending SignalMediaLibraryQueryUpdate.

5.3.20 SetMediaLibraryProgress

Set the indexing progress of the Media Library.

Syntax

```
CinemoError SetMediaLibraryStatus(  
    [in] const char * MediaLibraryUniqueIdentifier,  
    [in] uint32 progress  
) ;
```

Parameters

- **MediaLibraryUniqueIdentifier**

Media library ID.

- **progress**

Progress of the indexing process.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method shall be called from implementations of the [ICinemoIAPMediaLibraryAccess](#) interface to inform Cinemo about progress of the indexing process of iAP2 volumes. Based on these values Cinemo will decide which device browse functionality is available. Currently only a progress value of 100 has a functionality in Cinemo.

[SetMediaLibraryProgress\(\)](#) shall only be called while a device is mounted.

Only iAP2 devices support SetMediaLibraryProgress.

5.3.21 SendPlayMediaLibraryCommand

Send special play commands using Media Library information.

Syntax

```
CinemoError SendPlayMediaLibraryCommand(
    [in] const char * MediaLibraryUniqueIdentifier,
    [in] CINEMO_IAP_PLAYMEDIALIBRARY_COMMAND cmd,
    [in] CinemoIAP2UI32 starting_index,
    [in] CinemoIAP2UI64 starting_persistent_identifier,
    [in] CinemoIAP2UI64 collection_persistent_identifier,
    [in] CinemoIAP2UI64Array persistent_identifiers
);
```

Parameters

- **MediaLibraryUniqueIdentifier**

Media library ID.

- **cmd**

CINEMO_IAP_PLAYMEDIALIBRARY_COMMAND enum.

- **starting_index**

CollectionStartingIndex parameter.

- **starting_persistent_identifier**

StartingMediaPersistentIdentifier parameter.

- **collection_persistent_identifier**

CollectionPersistentIdentifier parameter.

- **persistent_identifiers**

ItemsPersistentIdentifiers parameter.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Using this method it is possible to directly send iAP2 commands related to the Media Library Access interface of the Apple device. When using Cinemo Media Management it is possible to send these commands directly from the native iAP2 device browse hierarchy and this method should not be used.

Depending on the value of the *cmd* field, different commands on the iAP2 link will be called. While all commands require a valid *MediaLibraryUniqueIdentifier* the other parameters are explained in the following table.

command	iAP command	starting_index	starting_persistent_identifier	collection_persistent_identifier	persistent_identifiers
ITEMS	<i>PlayMediaLibraryItems</i>	optional	no	no	required
COLLECTION_PLAYLIST	<i>PlayMediaLibraryCollection</i>	optional ¹	optional ¹	required	no
COLLECTION_ARTIST	<i>PlayMediaLibraryCollection</i>	no	optional	required	no
COLLECTION_ALBUM	<i>PlayMediaLibraryCollection</i>	no	optional	required	no
COLLECTION_ALBUMARTIST	<i>PlayMediaLibraryCollection</i>	no	optional	required	no
COLLECTION_GENRE	<i>PlayMediaLibraryCollection</i>	no	optional	required	no
COLLECTION_COMPOSER	<i>PlayMediaLibraryCollection</i>	no	optional	required	no
SPECIAL	<i>PlayMediaLibrarySpecial</i>	no	optional	no	no

¹ mutually exclusive

5.3.22 StartPowerUpdates

Request the Apple device to start sending power update messages.

Syntax

```
CinemoError StartPowerUpdates (
    [in] const CinemoIAPPowerUpdate& update
);
```

Parameters

- **update**

Power Update data structure.

Return value

If the method succeeds, it returns *CinemoNoError*. If this method is called on an instance having no [ICinemolAPPowerUpdate](#) assigned at the [Open\(\)](#) call it will return *CinemoErrorState*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling this method, all power update messages from the Apple device will be reported on the [ICinemolAPPowerUpdates](#) interface passed to the [Open\(\)](#) method.

All fields which are enabled in the [CinemolAPPowerUpdate](#) parameter will be requested from the Apple device.

5.3.23 StopPowerUpdates

Request the Apple device to stop sending power update messages.

Syntax

```
CinemoError StopPowerUpdates();
```

Return value

If the method succeeds, it returns *CinemoNoError*. If this method is called on an instance having no [ICinemoAPPowerUpdate](#) assigned at the [Open\(\)](#) call it will return *CinemoErrorState*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling this method, power update messages will no longer be announced on the [ICinemoAPPowerUpdates](#) interface passed to the [Open\(\)](#) method.

5.3.24 SendPowerSourceUpdate

Send a power source update message to an Apple device.

Syntax

```
CinemoError SendPowerSourceUpdate (
    [in] const CinemoIAPPowerSourceUpdate& update
);
```

Parameters

- **update**

Power Source Update data structure.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

The *CinemoIAPPowerSourceUpdate* structure is defined as follows:

```
typedef struct {
    CinemoIAP2UI16 AvailableCurrentForDevice;
    CinemoIAP2Bool DeviceBatteryShouldChargeIfPowerIsPresent;
} CinemoIAPPowerSourceUpdate;
```

5.3.25 StartCallStateUpdates

Inform the Apple device to start sending communications call state updates.

Syntax

```
CinemoError StartCallStateUpdates(  
    [in] uint32 flags  
) ;
```

Parameters

- **flags**

enable optional fields of the [ICinemolAPCommunications::CallStateUpdate\(\)](#) message according the bits set in flags. The bits are defined in the CINEMO_IAP_COMMUNICATIONS_CALLSTATE_UPDATES_FLAGS enumeration.

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemolAP](#). If at the time of call no [ICinemolAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling this method, all new and ongoing phone calls will be announced on the [ICinemolAPCommunications](#) interface passed to the [Open\(\)](#) method. This will result in calls to [ICinemolAPCommunications::CallStateUpdate\(\)](#).

5.3.26 StopCallStateUpdates

Inform the Apple device to stop sending communications call state updates.

Syntax

```
CinemoError StopCallStateUpdates();
```

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#). If at the time of call no [ICinemoIAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling this method, phone calls will no longer be announced on the [ICinemoIAPCommunications](#) interface passed to the [Open\(\)](#) method.

5.3.27 StartCommunicationsUpdates

Inform the Apple device to start sending communications status update messages.

Syntax

```
CinemoError StartCommunicationsUpdates(  
    [in] const CinemoIAPCommunicationsUpdate & select  
) ;
```

Parameters

- **update**

Structure for enabling CinemoIAPCommunicationsUpdate parameters.

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#). If at the time of call no [ICinemolAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling this method, all status changes of the Apple device will be announced on the [ICinemolAPCommunications](#) interface passed to the [Open\(\)](#) method.

The input parameter update will be evaluated to determine the desired output parameters of [ICinemolAPCommunications::CommunicationsUpdate\(\)](#). Each parameter which is set to valid in this call, will be present in the telephony update received via the [ICinemolAPCommunications](#) interface.

5.3.28 StopCommunicationsUpdates

Inform the Apple device to stop sending communications status update messages.

Syntax

```
CinemoError StopCommunicationsUpdates();
```

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemolAP](#). If at the time of call no [ICinemolAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling this method, any status changes of the Apple device will no longer be announced on the [ICinemolAPCommunications](#) interface passed to the [Open\(\)](#) method.

5.3.29 StartListUpdates

Inform the Apple device to start sending communications list update messages.

Syntax

```
CinemoError StartListUpdates(
    [in] const CinemoIAPCommunicationsListRequest & request
);
```

Parameters

- **update**

Structure for enabling CinemoIAPCommunicationsListUpdate parameters.

The CinemoIAPCommunicationsListRequest structure is defined as follows:

```
typedef struct CinemoIAPCommunicationsListRequest {
    bool RequestRecentsList;
    uint16 RecentsListMax;
    bool RecentsListCombine;
    CinemoIAP2RecentsListItem RecentsList;
    bool RequestFavoritesList;
    uint16 FavoritesListMax;
    CinemoIAP2FavoritesListItem FavoritesList;
} CinemoIAPCommunicationsListRequest;
```

Set the appropriate values and select the optional fields to customize the information sent in the [ICinemoIAPCommunications::ListUpdate\(\)](#) callback. Using the value zero (0) in the max fields implies no limit.

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#). If at the time of call no [ICinemoIAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling this method, all status changes of the Apple device will be announced on the [ICinemoIAPCommunications](#) interface passed to the [Open\(\)](#) method.

The input parameter update will be evaluated to determine the desired output parameters of [ICinemoIAPCommunications::ListUpdate\(\)](#). Each parameter which is set to valid in this call, will be present in the telephony update received via the [ICinemoIAPCommunications](#) interface.

5.3.30 StopListUpdates

Inform the Apple device to stop sending communications list update messages.

Syntax

```
CinemoError StopListUpdates();
```

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemolAP](#). If at the time of call no [ICinemolAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

Remarks

After calling this method, any status changes of the Apple device will no longer be announced on the [ICinemolAPCommunications](#) interface passed to the [Open\(\)](#) method.

5.3.31 InitiateCall

Initiate a phone call via the connected Apple device.

Syntax

```
CinemoError InitiateCall(  
    [in] CINEMO_IAP_COMMUNICATIONS_CALLTYPE type,  
    [in] const char *                                destinationID,  
    [in] CINEMO_IAP_COMMUNICATIONS_SERVICE   service,  
    [in] const char *                                addressBookID  
) ;
```

Parameters

- **type**

type of call: destination, voicemail, redial.

- **destinationID**

number to call, valid only if type == 'destination'.

- **service**

service to use, valid only if type == 'destination'.

- **addressBookID**

addressbookID, valid only if type == 'destination'.

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#). If at the time of call no [ICinemoIAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

5.3.32 AcceptCall

Accept an incoming call announced via [ICinemoIAPCommunications::CallStateUpdate\(\)](#).

Syntax

```
CinemoError AcceptCall(  
    [in] CINEMO_IAP_COMMUNICATIONS_ACCEPTCALLACTION acceptAction,  
    [in] const char *                           callUUID  
) ;
```

Parameters

- **acceptAction**

how to accept a call.

- **callUUID**

call identification.

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#).

If at the time of call no [ICinemoIAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

5.3.33 EndCall

Terminate or deny a call on the connected Apple device.

Syntax

```
CinemoError EndCall(  
    [in] CINEMO_IAP_COMMUNICATIONS_ENDCALLACTION endAction,  
    [in] const char *                           callUUID  
) = 0;
```

Parameters

- **endAction**
how to end a call
- **callUUID**
call identification

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#). If at the time of call no [ICinemoIAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

5.3.34 SwapCalls

Swap phone calls on a connected Apple device.

Syntax

```
CinemoError SwapCalls();
```

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemolAP](#).

If at the time of call no [ICinemolAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

5.3.35 MergeCalls

Merge phone calls on a connected Apple device.

Syntax

```
CinemoError MergeCalls();
```

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemolAP](#). If at the time of call no [ICinemolAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

5.3.36 HoldStatusUpdate

Update the hold status of a call.

Syntax

```
CinemoError HoldStatusUpdate(  
    [in] bool      holdStatus,  
    [in] const char * callUUID  
) ;
```

Parameters

- **holdStatus**

hold status: on or off

- **callUUID**

optional call identification, can be NULL

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#).

If at the time of call no [ICinemoIAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

5.3.37 MuteStatusUpdate

Update the mute status of a call.

Syntax

```
CinemoError MuteStatusUpdate(  
    [in] bool muteStatus  
) ;
```

Parameters

- **muteStatus**

mute status: on or off

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#). If at the time of call no [ICinemoIAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

5.3.38 SendDTMF

Send DTMF tone through existing call on the Apple device.

Syntax

```
CinemoError SendDTMF (
    [in] CINEMO_IAP_COMMUNICATIONS_DTMFTONE tone,
    [in] const char *                      callUUID
```

Parameters

- **tone**
DTMF tone value
- **callUUID**
optional call identification, can be NULL

Return value

If the method succeeds, it returns *CinemoNoError*. This method can be called on any instance of [ICinemoIAP](#). If at the time of call no [ICinemoIAPCommunications](#) instance is known to the Cinemo Media Engine, then this call will return with *CinemoErrorNoSuchObject*. Otherwise, it returns a *CinemoError* code.

5.3.39 SendVehicleStatus

Send Vehicle Status messages to an Apple device.

Syntax

```
CinemoError SendVehicleStatus (
    [in] const CinemoIAPVehicleStatus& status
);
```

Parameters

- **status**

Vehicle status data structure.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

In order to provide vehicle status data to an Apple device, an instance of the *ICinemoIAPVehicleStatus* interface needs to be passed to the [Open\(\)](#) method. Once the start method of this interface has been called, [SendVehicleStatus\(\)](#) needs to be called whenever the vehicle status changes as specified in the Apple MFi Spec.

[SendVehicleStatus\(\)](#) will return with *CinemoErrorDomain* if it is called outside of a vehicle status session started by the Apple device.

5.3.40 BluetoothComponentInformation

Inform the Apple device about the status of the Bluetooth components listed in the identification information passed at the Open call.

Syntax

```
CinemoError BluetoothComponentInformation(  
    [in] int num_status,  
    [in] const CinemoIAPBluetoothStatus status[]  
) ;
```

Parameters

- **num_status**

Number of elements in the status array passed as second parameter.

- **status**

An array of CinemoIAPBluetoothStatus elements.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

For each Bluetooth component which has been added to the meta pool used for the identification with the Apple device an initial status of the Bluetooth component must be sent to the Apple device. This function must be sent at most 500 ms after the identification succeeded. For iAP1 the fields deviceStatus, deviceType and deviceClass must be set. For iAP2 only the validity of the services bitfield must be set. It doesn't hurt if both types of fields are set regardless of the type of iAP connection. In every case the field component_id must match to the Bluetooth component identifier used in the identification.

5.3.41 StartBluetoothConnectionUpdates

Requests from the Apple device to start sending bluetooth connection updates.

Syntax

```
CinemoError StartBluetoothConnectionUpdates();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the Apple device is connected via the iAP2 protocol and as soon as this function has been called it starts to send BluetoothConnectionUpdate messages. They will be reported via the [ICinemoIAPBluetoothUpdates::BluetoothConnectionUpdate\(\)](#) method. For iAP1 the updates will start as soon as [BluetoothComponentInformation\(\)](#) has been called. Calling this function when iAP1 is used has no effect and it returns with *CinemoNoError*.

5.3.42 StopBluetoothConnectionUpdates

Requests from the Apple device to stop sending bluetooth connection updates.

Syntax

```
CinemoError StopBluetoothConnectionUpdates();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

If the Apple device is connected via the iAP2 protocol and as soon as this function has been called it stops to send BluetoothConnectionUpdate messages. For iAP1 the updates can not be stopped. Calling this function when iAP1 is used has no effect and it returns with *CinemoNoError*.

5.3.43 RequestWiFiInformation

Send a request WiFi information message to the Apple device to request the corresponding information from the Apple device.

Syntax

```
CinemoError RequestWiFiInformation();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

When this method is successful and processed by the Apple device it might answer with [ICinemoAPWiFilInfoSharing::WiFilInformation\(\)](#) call.

5.3.44 AccessoryWiFiConfigurationInformation

Report the accessory WiFi configuration to the Apple Device.

Syntax

```
CinemoError AccessoryWiFiConfigurationInformation(  
    [in] const CinemoIAPAccessoryWifiConfigInfo & info  
) ;
```

Parameters

- **info**

A filled instance of `CinemoIAPAccessoryWifiConfigInfo`.

The `CinemoIAPAccessoryWifiConfigInfo` structure is defined as follows:

```
typedef struct {  
    const char * WiFiSSID;  
    CinemoIAP2Utf8 Passphrase;  
    CinemoIAP2Enum SecurityType;  
    CinemoIAP2UI08 Channel;  
} CinemoIAPAccessoryWifiConfigInfo;
```

The structure corresponds to the information requested by the Apple device according to Accessory Interface Specification R27.

Return value

If the method succeeds, it returns `CinemoNoError`. Otherwise, it returns a `CinemoError` code.

Remarks

Use this method to answer to a request for accessory WiFi information from the Apple device received via a call to [|CinemoAPWiFilfoSharing::RequestAccessoryWiFiConfigurationInformation\(\)](#).

5.3.45 StartAssistiveTouch

Enables assistive touch feature of the Apple Device

Syntax

```
CinemoError StartAssistiveTouch();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

5.3.46 StopAssistiveTouch

Disables assistive touch feature of the Apple Device

Syntax

```
CinemoError StopAssistiveTouch();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

5.3.47 StartAssistiveTouchInformation

Enables assistive touch information from the Apple Device

Syntax

```
CinemoError StartAssistiveTouchInformation();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

5.3.48 StopAssistiveTouchInformation

Disables assistive touch information from the Apple Device

Syntax

```
CinemoError StopAssistiveTouchInformation();
```

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

5.3.49 OOBTParingAccessoryInformation

Transfer bluetooth accessory information to the Apple device.

Syntax

```
CinemoError OOBTPairingAccessoryInformation(  
    [in] uint16 btTransportComponentId,  
    [in] uint32 deviceClass  
) ;
```

Parameters

- **btTransportComponentId**

The id of the bluetooth component used in identification.

- **deviceClass**

The bluetooth device class of the accessory.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

5.3.50 OOBTPairingCompletionInformation

Report result of storing pairing link key information to the Apple device.

Syntax

```
CinemoError OOBTPairingCompletionInformation(  
    [in] uint8 resultCode  
) ;
```

Parameters

- **resultCode**

a result of '0' indicates success.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

5.3.51 StartRouteGuidanceUpdates

Start Route Guidance feature on the Apple device.

Syntax

```
CinemoError StartRouteGuidanceUpdates (
    [in] CinemoIAP2UI16Array component_ids
);
```

Parameters

- **component_ids**

IDs of the components to start.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Using the Route Guidance feature of an Apple device requires an *ICinemolAPRouteGuidance* interface in the *CinemolAPConfig* object of the [Open\(\)](#) method as well as an additional identification metapool entry (*CINEMO_METANAME_IAP_RGD_...*).

The *component_ids* correspond to the identification metapool index value of each component. If left blank, all components will be started.

After calling this method Route Guidance messages will be sent on the [*ICinemolAPRouteGuidance*](#) interface when they are received from the Apple device.

This feature is only available on devices running iOS 10 or later.

5.3.52 StopRouteGuidanceUpdates

Stop Route Guidance feature on the Apple device.

Syntax

```
CinemoError StopRouteGuidanceUpdates (
    [in] CinemoIAP2UI16Array component_ids
);
```

Parameters

- **component_ids**

IDs of the components to stop.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

Using the Route Guidance feature of an Apple device requires an *ICinemolAPRouteGuidance* in the *CinemolAPConfig* object of the [Open\(\)](#) method as well as an additional identification metapool entry (*CINEMO_METANAME_IAP_RGD_...*).

The *component_ids* correspond to the identification metapool index value of each component. If left blank, all components will be stopped.

After calling this method Route Guidance messages will no longer be sent on the [ICinemolAPRouteGuidance](#) interface.

This feature is only available on devices running iOS 10 or later.

5.4 ICinemoIAPEAPAnnounce

The [ICinemoIAPEAPAnnounce](#) interface is a pure callback interface, which needs to be implemented by the customer of the Cinemo IAP API. Instances of the implementation of the interface will be used in the Cinemo Media Engine to announce activity on iAP external accessory protocol (EAP) sessions to the customer. The activities are starting and stopping of an EAP session and when data coming from the Apple device is available. Please note that this interface is not thread-safe.

Multiple instances of this interface can be added with subsequent [ICinemoIP::Open\(\)](#) calls. All will be accumulated. If at the time of the call to [ICinemoIP::Open\(\)](#) initiated and pending EAP sessions are available, the [EAPSessionInitiated\(\)](#) function of the newly added instance is called for each EAP session, just in case it would like to accept one of them. The [ICinemoIAPEAPAnnounce](#) instance will be removed from the pool of accumulated instances when [ICinemoIP::Close\(\)](#) is called or the instance of [ICinemoIP](#) is released. All associated EAP sessions will be closed too.

ICinemoIAPEAPAnnounce Methods

METHOD	DESCRIPTION
EAPSessionInitiated	Inform about the start of an EAP session
EndEAPSession	Inform about the end of an EAP session
EAPPacketArrived	Inform about arrival of a data packet for an EAP session

5.4.1 EAPSessionInitiated

This function will be called by Cinemo when a new EAP session is started from the Apple device.

Syntax

```
void EAPSessionInitiated(
    [in] uint16 session_id,
    [in] uint8 protocol_id
);
```

Parameters

- **session_id**

The session id number for the corresponding EAP connection.

- **protocol_id**

The protocol id used at device identification for a certain EA protocol for which the EAP session shall be established.

Remarks

With this function the creation of an EAP session is announced. The announcement consists of the *session_id* and the *protocol_id* of the session. The *session_id* is the value used to identify this EAP connection. It needs to be passed to all functions related to EAP functionality in the Cinemo API.

When an EAP session is announced it needs to be either *accepted* or *denied*. Until this has happened the EAP session is kept internally in Cinemo in *pending* state.

To define the session status a call to [ICinemoIP::EAPSessionStatus\(\)](#) must happen during the time the Apple device waits for acknowledgement or denial of the session. From Cinemos observation this time is about ten seconds, but the Apple Accessory Specification does not provide a value. After this time the session will be closed from the Apple device.

It is *not necessary* to call from within the execution time of this function the [ICinemoIP::EAPSessionStatus\(\)](#) function for setting the EAP session connection status (but it is possible).

If more than one instance of [ICinemoIAPEAPAnnounce](#) is registered through multiple instances of [ICinemoIP](#), then all instances available at session initiation time receive this call. Only one of them should then handle the status of the session. All other should do nothing about the session.

EAP sessions kept in *pending* state are being announced (again) on instances of [ICinemoIAPEAPAnnounce](#) registered after the session announcement was received from the Apple device and none of the instances of [ICinemoIAPEAPAnnounce](#) known at that time started handling the EAP session.

Thus this function can trigger the creation of an other thread or process to handle the EAP session communication. That instance should then call the [ICinemoIP::EAPSessionStatus\(\)](#) method after it received its own EAP session announcement.

5.4.2 EndEAPSession

This function will be called by the Cinemo Media engine when the Apple device requests the end of an EAP session or when the session was terminated through the call to [ICinemolAP::EAPSessionStatus\(session_id, CINEMO_IAP_EAP_SESSION_STATUS_CLOSE\)](#). This method will also be called when [ICinemolAP::Close\(\)](#) is called and active EAP sessions still exist.

Syntax

```
void EndEAPSession(  
    [in] uint16 session_id  
) ;
```

Parameters

- **session_id**

The session id number for the corresponding EAP connection.

Remarks

An EAP session can be closed only by the Apple device. If this is the case Cinemo media engine calls this function to inform about this fact. The call happens only on the [ICinemolAPEAPAnnounce](#) instance associated with the instance of [ICinemolAP](#) where the [ICinemolAP::EAPSessionStatus\(session_id, CINEMO_IAP_EAP_SESSION_STATUS_OK\)](#) has been called.

A possibly blocked call to [ICinemolAP::EAPSessionRecv\(\)](#) will be unblocked.

Do not call [ICinemolAP::EAPSessionStatus\(session_id, CINEMO_IAP_EAP_SESSION_STATUS_CLOSE\)](#) from within this callback because all Cinemo internal resources have already been deallocated. The purpose of [ICinemolAP::EAPSessionStatus\(session_id, CINEMO_IAP_EAP_SESSION_STATUS_CLOSE\)](#) is to trigger the announcement of the end of the EAP session (i.E. this callback) and not to free resources.

5.4.3 EAPPacketArrived

This function will be called by the Cinemo Media engine when the Apple device has sent a packet of EAP session data to the accessory.

Syntax

```
void EAPPacketArrived(  
    [in] uint16 session_id  
    [in] uint32 numPackets  
) ;
```

Parameters

- **session_id**

The session id number for the corresponding EAP connection.

- **numPackets**

Number of available packets of data.

Remarks

This function allows to receive the data when it arrives on an EAP session at the point in time when it is available, without the need to wait blocking in the call to the [ICinemolAP::EAPSessionRecv\(\)](#) function. It is allowed to call [ICinemolAP::EAPSessionRecv\(\)](#) from within this function *numPackets* times. The *numPackets* gives the number of [ICinemolAP::EAPSessionRecv\(\)](#) calls which will retrieve packet data without blocking. Usually the value will be 1 (one).

In the case that an EAP session was for a longer duration in *pending* and EAP packet data arrived until the call to [ICinemolAP::EAPSessionStatus\(session_id, CINEMO_IAP_EAP_SESSION_STATUS_OK\)](#) was received, then the *numPackets* value might be bigger than one. The value of *numPackets* will not be zero. The call happens only on the [ICinemolAPEPAnnounce](#) instance associated with the instance of [ICinemolAP](#) where the [ICinemolAP::EAPSessionStatus\(session_id, CINEMO_IAP_EAP_SESSION_STATUS_OK\)](#) has been called to accept the session.

If after the call to [EAPPacketArrived\(\)](#) any data is still stored in the session a possibly blocked call to [ICinemolAP::EAPSessionRecv\(\)](#) will be unblocked.

5.5 ICinemolAPLocationInfo

To enable the location feature of Apple devices, it needs to be enabled during the identification process. Otherwise location information will never be requested by an Apple device. If location messages are sent by an Apple device, Cinemo Media Engine will pass them to an ICinemolAPLocationInfo interface passed in the CinemolAPConfig object of the [ICinemolAP::Open\(\)](#) method.

ICinemolAPLocationInfo Methods

METHOD	DESCRIPTION
StartLocationInformation	Inform about the start of a location information session.
GPRMCDatasStatusValuesNotification	Information about valid GPRMC status data values received.
StopLocationInformation	Inform about the end of a Location Information session.

5.5.1 StartLocationInformation

This function will be called by the Cinemo Media Engine when the Apple device requests to start sending location information messages.

Syntax

```
CinemoError StartLocationInformation(  
    [in] const int message_flags  
) ;
```

Parameters

- **message_flags**

A bitmask of valid CINEMO_IAP_LOCATION_MESSAGES for this session.

Return value

The return value of this function should always be CinemoNoError.

Remarks

As soon as this function is called, valid NMEA location messages need to be sent to the Apple device at regular time intervals. The message_flags parameter can be evaluated to see which kind of NMEA messages the Apple device can process. Usually GPGGA and GPRMC messages are recognized on all devices supporting location information.

5.5.2 GPRMCDataStatusValuesNotification

The Apple device sent an update for valid GPRMC status data values.

Syntax

```
CinemoError GPRMCDataStatusValuesNotification(
    [in] const int message_flags
);
```

Parameters

- **message_flags**

A bitmask of valid CINEMO_IAP_LOCATION_GPRMC_VALUE for this session.

Return value

The return value of this function should always be CinemoNoError.

Remarks

This callback provides a bitmask of valid GPRMC status data values. Only devices connected with the iAP2 protocol are able to provide this information. For iAP1 devices this callback will never be called.

5.5.3 StopLocationInformation

This function will be called by the Cinemo Media Engine when the Apple device requests to stop sending location information messages.

Syntax

```
CinemoError StopLocationInformation();
```

Return value

The return value of this function should always be CinemoNoError.

Remarks

As soon as this function is called, location messages need no longer be sent to the Apple device.

5.6 ICinemoAPMediaItemReceiver

The [ICinemoAPMediaItemReceiver](#) interface is an interface for passing CinemoAP2MediaItems back to Cinemo. It is used only if Cinemo Media Management is not available in the Cinemo SDK and can be ignored now.

ICinemoAPMediaItemReceiver Methods

METHOD	DESCRIPTION
OnCollectionQueryItemCount	Prepare the media item receiver for data
OnMediaItem	Return metadata for the requested items

5.6.1 OnCollectionQueryItemCount

Prepare the media item receiver for data.

Syntax

```
CinemoError OnCollectionQueryItemCount(
    [in] int count
);
```

Parameters

- **count**

Number of items to expect for the current query.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

This method will prepare the media item receiver for receiving a certain amount of data. Internally memory for this amount of items will be allocated when receiving the first item. If this method is not called a default amount of 64 items is assumed.

5.6.2 OnMediaItem

Return metadata for the requested items.

Syntax

```
CinemoError OnMediaItem(  
    [in] const CinemoIAP2MediaItem & item  
) ;
```

Parameters

- **item**

CinemoIAP2MediaItem containing all available metadata.

Return value

If the method succeeds, it returns *CinemoNoError*. Otherwise, it returns a *CinemoError* code.

Remarks

With this interface it is possible to pass results of [ICinemoIAPMediaLibraryAccess::GetMediaItem\(\)](#) or [ICinemoIAPMediaLibraryAccess::GetCollectionListing\(\)](#) back to Cinemo Media Engine.

To return more than one item, the method needs to be called multiple times.

5.7 ICinemoIAPMediaLibraryAccess

The [ICinemoIAPMediaLibraryAccess](#) interface is a pure callback interface, which needs to be implemented by the customer of the Cinemo IAP API only if Cinemo Media Management is not available within the Cinemo SDK.

Cinemo Media Management uses an instance of this interface to initiate media library updates from the Apple device as soon the device is mounted in CinemoMM. It will perform the call to [ICinemoIAP::StartMediaLibraryInformation\(\)](#) itself. Thus there is no need to take any action regarding this interface, because in the configuration of the Cinemo SDK Cinemo Media Management is enabled.

ICinemoIAPMediaLibraryAccess Methods

METHOD	DESCRIPTION
OnMediaLibraryInformation	New MediaLibraryInformation has arrived
OnMediaLibraryUpdate	New MediaLibraryUpdates have arrived
GetMediaItem	Used to query track information from the persistent storage
GetCollectionListing	Used for general metadata queries
OnArtwork	Artwork data is available
OnStatus	Status information is available
StopWatching	Stop calculating updates for an active query.

5.7.1 OnMediaLibraryInformation

Called if new MediaLibraryInformation has arrived from the Apple device.

Syntax

```
CinemoError OnMediaLibraryInformation(  
    [in] const CinemoIAP2MediaLibraryInfo & libInfo  
) ;
```

Parameters

- **libInfo**

Structure containing media library information.

Return value

The return value of this method is ignored.

Remarks

After calling [ICinemolAP::StartMediaLibraryInformation\(\)](#) and registering an ICinemolAPMediaLibraryAccess interface, this method will be called to inform about new information about media libraries on the Apple device.

The CinemolAP2MediaLibraryInfo structure is defined as follows:

```
typedef struct {  
    CinemoIAP2Utf8 Name;  
    CinemoIAP2Utf8 UniqueIdentifier;  
    CinemoIAP2Enum Type;  
} CinemoIAP2MediaLibraryInfo;
```

The Type argument corresponds to the iAP2 MediaLibraryType enum, where currently a value of 0 means local device library and a value of 2 iTunes Radio library.

5.7.2 OnMediaLibraryUpdate

Called if new MediaLibraryUpdates have arrived from the Apple device.

Syntax

```
CinemoError OnMediaLibraryUpdate(
    [in] const CinemoIAP2MediaLibraryUpdateInfo & libUpdate
);
```

Parameters

- **libUpdate**

Structure containing media library update information.

Return value

The return value of this method is ignored.

Remarks

All media library updates from the Apple device will be passed on to this method.

The CinemoIAP2MediaLibraryUpdateInfo structure is defined as follows:

```
typedef struct {
    CinemoIAP2Utf8 UniqueIdentifier;
    CinemoIAP2Utf8 Revision;
    CinemoIAP2MediaItemArray Items;
    CinemoIAP2MediaPlaylistArray Playlists;
    CinemoIAP2UI64Array DeleteItems;
    CinemoIAP2UI64Array DeletePlaylists;
    CinemoIAP2Void Reset;
    CinemoIAP2UI08 UpdateProgress;
    CinemoIAP2Bool IsHidingRemoteItems;
} CinemoIAP2MediaLibraryUpdateInfo;
```

a CinemoIAP2MediaItem is defined as:

```
typedef struct {
    CinemoIAP2UI64 PersistentIdentifier;
    CinemoIAP2Utf8 Title;
    CinemoIAP2Enum Type;
    CinemoIAP2UI08 Rating;
    CinemoIAP2UI32 PlaybackDurationInMilliseconds;
    CinemoIAP2UI64 AlbumPersistentIdentifier;
    CinemoIAP2Utf8 AlbumTitle;
    CinemoIAP2UI16 AlbumTrackNumber;
    CinemoIAP2UI16 AlbumTrackCount;
    CinemoIAP2UI16 AlbumDiscNumber;
    CinemoIAP2UI16 AlbumDiscCount;
    CinemoIAP2UI64 ArtistPersistentIdentifier;
}
```

```
CinemoIAP2Utf8 Artist;
CinemoIAP2UI64 AlbumArtistPersistentIdentifier;
CinemoIAP2Utf8 AlbumArtist;
CinemoIAP2UI64 GenrePersistentIdentifier;
CinemoIAP2Utf8 Genre;
CinemoIAP2UI64 ComposerPersistentIdentifier;
CinemoIAP2Utf8 Composer;
CinemoIAP2Bool IsPartOfCompilation;
CinemoIAP2Bool IsLikeSupported;
CinemoIAP2Bool IsBanSupported;
CinemoIAP2Bool IsLiked;
CinemoIAP2Bool IsBanned;
CinemoIAP2Bool IsResidentOnDevice;
CinemoIAP2Blob Artwork;
CinemoIAP2UI16 ChapterCount;
} CinemoIAP2MediaItem;
```

and a CinemoIAP2PlaylistItem as:

```
typedef struct {
    CinemoIAP2UI64 PersistentIdentifier;
    CinemoIAP2Utf8 Name;
    CinemoIAP2UI64 ParentPersistentIdentifier;
    CinemoIAP2Bool IsGeniusMix;
    CinemoIAP2Bool IsFolder;
    CinemoIAP2UI64Array ContainedMediaItemsUIDs;
    CinemoIAP2Bool IsiTunesRadio;
} CinemoIAP2MediaPlaylist;
```

5.7.3 GetMediaItem

Used to query track information from the persistent storage.

Syntax

```
CinemoError GetMediaItem(  
    [in] uint64 UID,  
    [in] ICinemolAPMediaItemReceiver * recv  
) ;
```

Parameters

- **UID**

PersistentIdentifier of the track to query.

- **recv**

ICinemolAPMediaItemReceiver interface for transmitting the result.

Return value

If the query is successful, this method should return *CinemoNoError*. If no track is found, it should return *CinemoErrorNoSuchObject*.

Remarks

This method will be called if metadata for a certain track is required. All available metadata need to be stored in an ICinemolAP2MediaItem and passed back via the [ICinemolAPMediaItemReceiver](#) interface to the Cinemo Media Engine.

5.7.4 GetCollectionListing

Used for general queries which may contain search criteria.

Syntax

```
CinemoError GetCollectionListing(
    [in] const CinemoIAPCollectionQuery & query,
    [in] ICinemoIAPMediaItemReceiver * recv,
    [in/out] uint32 & total_count
);
```

Parameters

- **query**

Input parameters for the query.

- **recv**

ICinemoIAPMediaItemReceiver interface for transmitting the results.

- **total_count**

Total number of items matching the query.

Return value

If the query is successful, this method should return *CinemoNoError*. If no matching item is found, it should return *CinemoErrorNoSuchObject*.

Remarks

This method will be used for general queries of the persistent storage. All items matching the search criteria need to be passed back one item at a time to the Cinemo Media Engine via the ICinemoIAPMediaItemReceiver interface.

The parameters for the search query are passed as a CinemoIAPCollectionQuery structure. It is defined as follows:

```
typedef struct {
    CINEMO_IAP_QUERYTYPE type;
    CinemoIAPCIDFilter filter;
    CINEMO_IAP_SORTFIELD sortfield;
    uint32 index;
    uint32 count;
    uint64 result_item_flags;
    uint32 queryID;
} CinemoIAPCollectionQuery;
```

where *CINEMO_IAP_QUERYTYPE* may be one of the following:

```
typedef enum {
    CINEMO_IAP_QUERYTYPE_PLAYLIST,
```

```
CINEMO_IAP_QUERYTYPE_ARTIST,
CINEMO_IAP_QUERYTYPE_ALBUM,
CINEMO_IAP_QUERYTYPE_ALBUMARTIST,
CINEMO_IAP_QUERYTYPE_GENRE,
CINEMO_IAP_QUERYTYPE_COMPOSER,
CINEMO_IAP_QUERYTYPE_TRACKS,
CINEMO_IAP_QUERYTYPE_AUDIOBOOK,
CINEMO_IAP_QUERYTYPE_PODCAST,
CINEMO_IAP_QUERYTYPE_GENIUS_MIXES,
CINEMO_IAP_QUERYTYPE_ITUNESU,
CINEMO_IAP_QUERYTYPE_ITUNESRADIO,
} CINEMO_IAP_QUERYTYPE;
```

and CINEMO_IAP_SORTFIELD:

```
typedef enum {
    CINEMO_IAP_SORTFIELD_NAME,
    CINEMO_IAP_SORTFIELD_INDEX,
    CINEMO_IAP_SORTFIELD_PLAYLISTORDER,
} CINEMO_IAP_SORTFIELD;
```

The type defines how the search result should be grouped. A type of *CINEMO_IAP_QUERYTYPE_ALBUM* should return items which are grouped by album.

Filtering options may be set in the CinemoAPCIDFilter structure, which is defined as:

```
typedef struct {
    CinemoIAP2UI64 playlist;
    CinemoIAP2UI64 artist;
    CinemoIAP2UI64 album;
    CinemoIAP2UI64 albumartist;
    CinemoIAP2UI64 genre;
    CinemoIAP2UI64 composer;
} CinemoAPCIDFilter;
```

If any of these values is set, all search results must match the collection identifier specified in the corresponding variable.

The remaining fields in the CinemoAPCollectionQuery structure denote the search order of the results as well as the result count and index. An index and count of *UINT32_MAX* requests all items which match these criteria.

The total_count argument needs to be set to the total number of search results, regardless of the values of count and index. This is needed to ensure the proper functionality of the ICinemoPlaylist.

To reduce the number of metadata transferred the *result_item_flags* parameter is used. It contains a bitmask of *CINEMO_IAP_MEDIAINFO_FLAGS* and defines the entries in a CinemoAP2MediaItem which should be returned via the ICinemolAPMediaItemReceiver interface.

All queries will contain a unique *queryID*. Each query shall be active until [StopWatching\(\)](#) is called for this particular ID.

5.7.5 OnArtwork

Artwork data is available for a certain track.

Syntax

```
CinemoError OnArtwork(  
    [in] uint64 UID,  
    [in] ICinemoBlob * pdata,  
    [in] const CinemoIAP2UI32 & timestamp  
) ;
```

Parameters

- **UID**

UID of the corresponding track.

- **pdata**

Artwork image data.

- **timestamp**

Timestamp of the artwork (if available).

Return value

The return value of this method is ignored.

Remarks

Currently the iAP2 protocol does not transfer artwork when using MediaLibraryUpdates. Artwork is only available for “Now Playing” updates, which are usually processed by Cinemo internally. If a track is played on the device which contains artwork, Cinemo will pass this information on to this method. If the “Now Playing” update also contains a timestamp, it will also be available.

5.7.6 OnStatus

Status information is available for the Media Library Access interface.

Syntax

```
CinemoError OnStatus(  
    [in] uint32 status  
) ;
```

Parameters

- **status**
Bitmask of CINEMO_IAP_MEDIALIB_STATUS.

Return value

The return value of this method is ignored.

Remarks

This method will be called every time the status of the Media Library Access interface changes. It is tied to the [ICinemolAP::SetMediaLibraryStatus\(\)](#) and forwards the information provided in that call. It is up to the implementation to handle this information properly.

5.7.7 StopWatching

Stop calculating updates for an active query.

Syntax

```
CinemoError StopWatching(  
    [in] uint32 queryID  
) ;
```

Parameters

- **queryID**

ID of the query which is closed.

Return value

The return value of this method is ignored.

Remarks

If an active query is closed, this method will be called to inform the Media Library to stop calculating dynamic updates for this query.

5.8 ICinemolAPPowerUpdates

Power updates from an Apple device can be enabled to receive information about current and battery statistics. These messages need to be specified in the identification process and will not be available otherwise. If power messages are sent by an Apple device, Cinemo Media Engine will pass them to an ICinemolAPPowerUpdates interface passed in the CinemoIAPConfig object of the [ICinemolAP::Open\(\)](#) method.

ICinemolAPPowerUpdates Methods

METHOD	DESCRIPTION
PowerUpdate	Inform about a power update from the device.

5.8.1 PowerUpdate

This function will be called by the Cinemo Media Engine when the Apple device sends a power update message.

Syntax

```
CinemoError PowerUpdate(  
    [in] const CinemoIAPPowerUpdate& update  
) ;
```

Parameters

- **update**

A CinemoIAPPowerUpdate structure.

Return value

The return value of this function should always be CinemoNoError.

Remarks

The CinemoIAPPowerUpdate structure is defined as follows:

```
typedef struct {  
    CinemoIAP2UI16      MaxCurrentDrawnFromAccessory;  
    CinemoIAP2Bool     DeviceBatteryWillChargeIfPowerIsPresent;  
    CinemoIAP2Enum     AccessoryPowerMode;  
    CinemoIAP2Bool     IsExternalChargerConnected;  
    CinemoIAP2Enum     BatteryChargingState;  
    CinemoIAP2UI16      BatteryChargeLevel;  
} CinemoIAPPowerUpdate;
```

Both enums, AccessoryPowerMode and BatteryChargingState, are mapped to the corresponding enums of the Apple MFi Spec.

5.9 ICinemolAPCommunications

To enable the communications feature of Apple devices, it needs to be enabled during the identification process. If communications information is received from the Apple device, Cinemo Media Engine will pass it to an [ICinemolAPCommunications](#) instance passed in the CinemolAPConfig object of the [ICinemolAP::Open\(\)](#) method.

ICinemolAPCommunications Methods

METHOD	DESCRIPTION
CallStateUpdate	Inform about phone calls.
CommunicationsUpdate	Inform about a change in the Apple devices communications status.
ListUpdate	Inform about a change in the Apple devices communications list status.

5.9.1 CallStateUpdate

This function will be called by the Cinemo Media Engine when the Apple device sends a call state update message.

Syntax

```
CinemoError CallStateUpdate(
    [in] const CinemoIAPCommunicationsCallState& update
);
```

Parameters

- update**

Call state information update structure.

Return value

The return value of this function should always be CinemoNoError.

Remarks

This function will be called only if call status updates have been requested from the Apple device, by using [ICinemolAP::StartCallStateUpdates\(\)](#). With a call to [ICinemolAP::StopCallStateUpdates\(\)](#) call status updates can be disabled again.

The CinemolAPCommunicationsCallState structure is defined as follows:

```
typedef struct CinemoIAPCommunicationsCallState {
    CinemoIAP2Utf8 RemoteID;
    CinemoIAP2Utf8 DisplayName;
    CinemoIAP2Enum Status;
    CinemoIAP2Enum StatusLegacy;
    CinemoIAP2Enum Direction;
    CinemoIAP2Enum DirectionLegacy;
    CinemoIAP2Utf8 CallUUID;
    CinemoIAP2Utf8 AddressBookID;
    CinemoIAP2Utf8 Label;
    CinemoIAP2Enum Service;
    CinemoIAP2Bool IsConferenced;
    CinemoIAP2UI08 ConferenceGroup;
    CinemoIAP2Enum DisconnectReason;
    CinemoIAP2UI64 StartTimestamp;
} CinemoIAPCommunicationsCallState;
```

The enums Status, StatusLegacy, Direction, DirectionLegacy, Service and DisconnectReason are mapped to the corresponding enums of the Apple MFi Spec. Please be aware that only one of Status or StatusLegacy fields will have the member 'valid' set to true and never both. Also the enumeration values (in the member field 'value') differ depending on which of the two fields are set to valid. This applies in analog way to the Direction and DirectionLegacy fields.

5.9.2 CommunicationsUpdate

This function will be called by the Cinemo Media Engine when the Apple device sends a communications update message.

Syntax

```
CinemoError CommunicationsUpdate(
    [in] const CinemoIAPCommunicationsUpdate& update
);
```

Parameters

- **update**

Communications update state structure.

Return value

The return value of this function should always be CinemoNoError.

Remarks

This function will be called only if communications updates have been requested from the Apple device, by using [ICinemolAP::StartCommunicationsUpdates\(\)](#). With a call to [ICinemolAP::StopCommunicationsUpdates\(\)](#) communications updates can be disabled again.

The CinemolAPCommunicationsUpdate structure is defined as follows:

```
typedef struct CinemoIAPCommunicationsUpdate {
    CinemoIAP2Enum SignalStrength;
    CinemoIAP2Enum RegistrationStatus;
    CinemoIAP2Bool AirplaneModeStatus;
    CinemoIAP2Utf8 CarrierName;
    CinemoIAP2Bool CellularSupported;
    CinemoIAP2Bool TelephonyEnabled;
    CinemoIAP2Bool FaceTimeAudioEnabled;
    CinemoIAP2Bool FaceTimeVideoEnabled;
    CinemoIAP2Bool MuteStatus;
    CinemoIAP2UI08 CurrentCallCount;
    CinemoIAP2UI08 NewVoicemailCount;
    CinemoIAP2Bool InitiateCallAvailable;
    CinemoIAP2Bool EndAndAcceptAvailable;
    CinemoIAP2Bool HoldAndAcceptAvailable;
    CinemoIAP2Bool SwapAvailable;
    CinemoIAP2Bool MergeAvailable;
    CinemoIAP2Bool HoldAvailable;
} CinemoIAPCommunicationsUpdate;
```

Both enums SignalStrength and RegistrationStatus, are mapped to the corresponding enums of the Apple MFi Spec.

5.9.3 ListUpdate

This function will be called by the Cinemo Media Engine when the Apple device sends a communications list update message.

Syntax

```
CinemoError ListUpdate(
    [in] const CinemoIAPCommunicationsListUpdate& update
);
```

Parameters

- update**

Communications list update state structure.

Return value

The return value of this function should always be CinemoNoError.

Remarks

This function will be called only if communications list updates have been requested from the Apple device, by using [ICinemolAP::StartListUpdates\(\)](#). With a call to [ICinemolAP::StopListUpdates\(\)](#) communications list updates can be disabled again.

The CinemolAPCommunicationsListUpdate structure is defined as follows:

```
typedef struct CinemoIAPCommunicationsListUpdate {
    CinemoIAP2Bool           RecentsListAvailable;
    CinemoIAP2RecentsListArray RecentsList;
    CinemoIAP2UI16            RecentsListCount;
    CinemoIAP2Bool           FavoritesListAvailable;
    CinemoIAP2FavoritesListArray FavoritesList;
    CinemoIAP2UI16            FavoritesListCount;
} CinemoIAPCommunicationsListUpdate;
```

The CinemolAP2RecentsListArray is defined as follows:

```
typedef struct CinemoIAP2RecentsListArray {
    int                  count;
    const CinemoIAP2RecentsListItem * value;
} CinemoIAP2RecentsListArray;
```

The count member specifies how many CinemolAP2RecentsListItem entries have been allocated at the place where the value parameter points to.

The CinemolAP2FavoritesListArray is defined as follows:

```
typedef struct CinemoIAP2FavoritesListArray {
    int                  count;
```

```
    const CinemoIAP2FavoritesListItem * value;
} CinemoIAP2FavoritesListArray;
```

The count member specifies how many CinemoIAP2FavoritesListItem entries have been allocated at the place where the value parameter points to.

The CinemoIAP2RecentsListItem structure is defined as follows:

```
typedef struct CinemoIAP2RecentsListItem {
    uint16                               Index;
    const char *                         RemoteID;
    const char *                         DisplayName;
    CinemoIAP2Utf8                      Label;
    CinemoIAP2Utf8                      AddressBookID;
    CINEMO_IAP_COMMUNICATIONS_SERVICE   Service;
    CINEMO_IAP_COMMUNICATIONS_RECENTLIST_TYPE Type;
    CinemoIAP2UI64                      UnixTimestamp;
    CinemoIAP2UI32                      Duration;
    uint8                                Occurrences;
} CinemoIAP2RecentsListItem;
```

The CinemoIAP2FavoritesListItem structure is defined as follows:

```
typedef struct CinemoIAP2FavoritesListItem {
    uint16                               Index;
    const char *                         RemoteID;
    const char *                         DisplayName;
    CinemoIAP2Utf8                      Label;
    CinemoIAP2Utf8                      AddressBookID;
    CINEMO_IAP_COMMUNICATIONS_SERVICE   Service;
} CinemoIAP2FavoritesListItem;
```

All the memory allocated for the fields lives only as long as the function is called. When the function returns Cinemo releases the data. Thus the caller receiving this data must deep copy it into his own data keeping.

5.10 ICinemolAPVehicleStatus

To enable the vehicle status feature of Apple devices, it needs to be enabled during the identification process. Otherwise vehicle information will never be requested by an Apple device. If vehicle status messages are sent by an Apple device, Cinemo Media Engine will pass them to an ICinemolAPVehicleStatus interface passed in the CinemolAPConfig object of the [ICinemolAP::Open\(\)](#) method.

ICinemolAPVehicleStatus Methods

METHOD	DESCRIPTION
StartVehicleStatus	Inform about the start of a vehicle status session.
StopVehicleStatus	Inform about the end of a vehicle status session.

5.10.1 StartVehicleStatus

This function will be called by the Cinemo Media Engine when the Apple device requests to start sending vehicle status messages.

Syntax

```
CinemoError StartVehicleStatus(  
    [in] const CinemoIAPVehicleStatus& status  
) ;
```

Parameters

- **status**

Vehicle status structure.

Return value

The return value of this function should always be CinemoNoError.

Remarks

As soon as this function is called, vehicle status data need to be sent to the Apple device at regular time intervals. All parameters, which are valid in the status input parameter can be processed by the Apple device.

5.10.2 StopVehicleStatus

This function will be called by the Cinemo Media Engine when the Apple device requests to stop sending vehicle status messages.

Syntax

```
CinemoError StopVehicleStatus();
```

Return value

The return value of this function should always be CinemoNoError.

Remarks

As soon as this function is called, vehicle status messages need no longer be sent to the Apple device.

5.11 ICinemolAPBluetoothUpdates

To be able to use the Bluetooth connection update feature of Apple devices, it needs to be enabled during the identification process. If Bluetooth connection update messages are sent by an Apple device, Cinemo Media Engine will pass them to an instance of the ICinemolAPBluetoothUpdates interface passed in the CinemolAPConfig object of the [ICinemolAP::Open\(\)](#) method.

ICinemolAPBluetoothUpdates Methods

METHOD	DESCRIPTION
<i>BluetoothConnectionUpdate</i>	Provide information about Bluetooth connection status of the Apple device.
<i>InitialBluetoothComponentInformationRequest</i>	Informs about sending the initial BluetoothComponentInformation.

5.11.1 BluetoothConnectionUpdate

This function will be called by the Cinemo Media Engine when the Apple device provides status information about Bluetooth connections.

Syntax

```
CinemoError BluetoothConnectionUpdate(  
    [in] const CinemoIAPBluetoothStatus& status  
) ;
```

Parameters

- **status**

Bluetooth connection status structure.

Return value

The return value of this function should always be CinemoNoError.

Remarks

When this function is called, the Apple devices announce a state change in one of the identified Bluetooth components. The field component_id of the passed CinemoIAPBluetoothStatus instance identifies the component which has changed. The other fields in the passed CinemoIAPBluetoothStatus instance may be set or not. Those set have the valid field set to true.

5.11.2 InitialBluetoothComponentInformationRequest

This callback informs about the requirement of sending the initial BluetoothComponentInformation.

Syntax

```
CinemoError InitialBluetoothComponentInformationRequest();
```

Return value

The return value of this function is not evaluated.

Remarks

This callback is only triggered for iAP1 devices.

The MFi Specification requires sending an initial BluetoothComponentInformation within 500 ms of setting up the connection with the iAP device. On certain systems it is possible that finishing [ICinemolAP::Open\(\)](#) takes more time than than the 500 ms allowed by the MFi Specification. Cinemo provides this callback to facilitate the sending of the initial BluetoothComponentInformation while [ICinemolAP::Open\(\)](#) is still running.

This callback will be called while [ICinemolAP::Open\(\)](#) is running and a [ICinemolAP::BluetoothComponentInformation\(\)](#) may be sent directly from within this callback with no additional thread required.

5.12 ICinemolAPWiFilInfoSharing

The WiFi info sharing is used to receive WiFi information from the Apple device or to provide accessory WiFi information for the Apple device. The corresponding iAP2 messages need to be specified during identification. When an implementation of this interface is provided in the CinemolAPConfig parameter of the [ICinemolAP::Open\(\)](#) method Cinemo Media Engine will call the corresponding method of this interface when a equivalent message has been received from the Apple device.

ICinemolAPWiFilInfoSharing Methods

METHOD	DESCRIPTION
<u>WiFiInformation</u>	Provided WiFi information parameters from the Apple device.
<u>RequestAccessoryWiFiConfigurationInformation</u>	The Apple device requests accessory WiFi configuration information.

5.12.1 WiFiInformation

This function will be called by the Cinemo Media Engine when the Apple device sends a WiFi information sharing message. To do so first the [ICinemolAP::RequestWiFiInformation\(\)](#) must be called.

Syntax

```
CinemoError WiFiInformation(  
    [in] const CinemoIAPWiFiInfo& info  
) ;
```

Parameters

- **info**

A CinemolAPWiFiInfo structure.

Return value

The return value of this function should always be CinemoNoError. The return value is not checked or used for anything in Cinemo Media Engine.

Remarks

The CinemolAPWiFiInfo structure is defined as follows:

```
typedef struct CinemoIAPWiFiInfo {  
    CINEMO_IAP_WIFI_REQUEST_STATUS RequestStatus;  
    CinemoIAP2Utf8             WiFiSSID;  
    CinemoIAP2Utf8             WiFiPassphrase;  
} CinemoIAPWiFiInfo;
```

The structure corresponds to the information provided by the Apple device according to Accessory Interface Protocol v2 (R23) specification.

5.12.2 RequestAccessoryWiFiConfigurationInformation

This function will be called by the Cinemo Media Engine when the Apple device sends a request accessory WiFi configuration information message. This will happen asynchronously during the life time of the [ICinemoIAP](#) instance.

Syntax

```
CinemoError RequestAccessoryWiFiConfigurationInformation();
```

Return value

The return value of this function should always be CinemoNoError. The return value is not checked or used for anything in Cinemo Media Engine.

Remarks

The called instance must answer this request with a call to [ICinemoIAP::AccessoryWiFiConfigurationInformation\(\)](#) either from within the requesting callback function or from a different thread.

5.13 ICinemolAPAssistiveTouch

The Assistive Touch interface can be used to generate iOS Multi-Touch gestures using one finger, a stylus or a compatible accessory for users with special needs. The corresponding iAP2 messages need to be specified during identification. When an implementation of this interface is provided in the CinemoIAPConfig parameter of the [ICinemolAP::Open\(\)](#) method Cinemo Media Engine will call the corresponding method of this interface when a equivalent message has been received from the Apple device.

ICinemolAPAssistiveTouch Methods

METHOD	DESCRIPTION
AssistiveTouchInformation	Status of assistive touch provided from the Apple device.

The features of this interface are a one to one mapping to the corresponding feature described in the Apple Accessory Interface Specification.

5.13.1 AssistiveTouchInformation

This function will be called by the Cinemo Media Engine when the Apple device sends a assistive touch information message.

Syntax

```
CinemoError AssistiveTouchInformation(  
    [in] bool enabled  
) ;
```

Parameters

- **enabled**
status of assistive touch

Return value

The return value of this function should always be CinemoNoError. The return value is not checked or used for anything in Cinemo Media Engine.

5.14 ICinemolAPOOBTPairing

The Out-of-Band Bluetooth Pairing feature enables accessories to perform Bluetooth pairing with an Apple device over a physical connection to enable smooth transitions between wired and wireless connections. The corresponding iAP2 messages need to be specified during identification. When an implementation of this interface is provided in the CinemoIPConfig parameter of the [ICinemolAP::Open\(\)](#) method Cinemo Media Engine will call the corresponding method of this interface when a equivalent message has been received from the Apple device.

ICinemolAPOOBTPairing Methods

METHOD	DESCRIPTION
<i>StartOOBTPairing</i>	Start the out of band Bluetooth pairing message exchange.
<i>OOBTPairingLinkKeyInformation</i>	Bluetooth pairing information from the Apple device.
<i>StopOOBTPairing</i>	Stop the out of band Bluetooth pairing message exchange.

The features of this interface are a one to one mapping to the corresponding feature described in the Apple Accessory Interface Specification.

5.14.1 StartOOBBTPairing

This function will be called by the Cinemo Media Engine when the Apple device requests to start out of band bluetooth pairing process.

Syntax

```
CinemoError StartOOBBTPairing();
```

Return value

The return value of this function should always be CinemoNoError.

Remarks

As soon as this function is called, calls to [ICinemolAP::OOBBTParingAccessoryInformation\(\)](#) and [ICinemolAP::OOBBTPairingCompletionInformation\(\)](#) shall inform the Apple device of the accessories bluetooth pairing information and status.

5.14.2 OOBTPairingLinkKeyInformation

Accept Bluetooth pairing information from the Apple device.

Syntax

```
CinemoError OOBTPairingLinkKeyInformation(  
    [in] CinemoIAP2Blob linkKey  
    [in] CinemoIAP2Blob appleDevMacAddr  
) ;
```

Parameters

- **linkKey**

Bluetooth connection link key.

- **appleDevMacAddr**

Bluetooth MAC address of the Apple device.

Return value

The return value of this function should always be CinemoNoError.

5.14.3 StopOOBBTPairing

This function will be called by the Cinemo Media Engine when the Apple device requests to stop out of band bluetooth pairing process.

Syntax

```
CinemoError StopOOBBTPairing();
```

Return value

The return value of this function should always be CinemoNoError.

5.15 ICinemolAPVoiceOver

VoiceOver is the screen reader feature built into Apple devices. Users who are blind or have low vision can interact with an Apple device. The corresponding iAP2 messages need to be specified during identification. When an implementation of this interface is provided in the CinemolAPConfig parameter of the [ICinemolAP::Open\(\)](#) method Cinemo Media Engine will call the corresponding method of this interface when a equivalent message has been received from the Apple device.

ICinemolAPVoiceOver Methods

METHOD	DESCRIPTION
VoiceOverUpdate	Voice Over update from the Apple device.
VoiceOverCursorUpdate	Voice over cursor update from the Apple device.

The features of this interface are a one to one mapping to the corresponding feature described in the Apple Accessory Interface Specification.

5.15.1 VoiceOverUpdate

This function will be called by the Cinemo Media Engine when the Apple device sends a voice over update message.

Syntax

```
CinemoError VoiceOverUpdate(  
    [in] CinemoIAP2UI08 speakingVolume,  
    [in] CinemoIAP2UI08 speakingRate,  
    [in] CinemoIAP2Bool enabled  
) ;
```

Parameters

- **speakingVolume**

0 = muted, 255 = loudest possible volume

- **speakingRate**

0 = off, 255 = fastest possible speed

- **enabled**

status of voice over

Return value

The return value of this function should always be CinemoNoError. The return value is not checked or used for anything in Cinemo Media Engine.

5.15.2 VoiceOverCursorUpdate

This function will be called by the Cinemo Media Engine when the Apple device sends a voice over cursor update message.

Syntax

```
CinemoError VoiceOverCursorUpdate(  
    [in] CinemoIAP2Utf8 label,  
    [in] CinemoIAP2Utf8 value,  
    [in] CinemoIAP2Utf8 hint,  
    [in] CinemoIAP2Blob traits  
) ;
```

Parameters

- **label**

label

- **value**

value

- **hint**

hint

- **traits**

traits: an array of uint16

Return value

The return value of this function should always be CinemoNoError. The return value is not checked or used for anything in Cinemo Media Engine.

5.16 ICinemolAPRouteGuidance

To be able to use the Route Guidance interface of Apple devices, it needs to be enabled during the identification process. If Route Guidance update messages are sent by an Apple device, Cinemo Media Engine will pass them to an instance of the ICinemolAPRouteGuidance interface passed in the CinemolAPConfig object of the [ICinemolAP::Open\(\)](#) method.

ICinemolAPRouteGuidance Methods

METHOD	DESCRIPTION
<i>RouteGuidanceUpdate</i>	Provide information about route guidance.
<i>RouteGuidanceManeuverUpdate</i>	Provide information about upcoming maneuvers.

5.16.1 RouteGuidanceUpdate

This function will be called by the Cinemo Media Engine when the Apple device provides Route Guidance Update messages.

Syntax

```
CinemoError RouteGuidanceUpdate(  
    [in] const CinemoIAPRouteGuidanceUpdate& update  
) ;
```

Parameters

- **update**

CinemolAPRouteGuidanceUpdate data structure.

Return value

The return value of this function should always be CinemoNoError.

Remarks

When this function is called, new Route Guidance information has been received from the Apple device.

5.16.2 RouteGuidanceManeuverUpdate

This function will be called by the Cinemo Media Engine when the Apple device provides Route Guidance Maneuver Update messages.

Syntax

```
CinemoError RouteGuidanceManeuverUpdate (
    [in] const CinemoIAPRouteGuidanceManeuverUpdate& update
);
```

Parameters

- **update**

CinemolIAPRouteGuidanceManeuverUpdate data structure.

Return value

The return value of this function should always be CinemoNoError.

Remarks

When this function is called, new maneuver information has been announced by the Apple device.

5.17 ICinemoCarPlay

The [ICinemoCarPlay](#) interface documented here exposes a full implementation of Apple's [CarPlay](#) client protocol. This consists of all IP-layer functionality including Apple's communication plugin, audio and video streaming, and light-weight mDNS (Bonjour) for service discovery.

ICinemoCarPlay Methods

METHOD	DESCRIPTION
<u>Start</u>	Start the CarPlay server.
<u>Stop</u>	Stop the CarPlay service.
<u>ForceKeyFrame</u>	Force a keyframe.
<u>RequestSiri</u>	Send a siri request
<u>SetLimitedUI</u>	Enable or disable the limited UI
<u>SetNightMode</u>	Enable or disable the night mode
<u>RequestUI</u>	Request a specific UI
<u>SetCallbacks</u>	Set callbacks for notification of client events.
<u>SetInfo</u>	Set client information.
<u>SetMainAudioParams</u>	Set audio device parameters for Main audio playback
<u>SetAlternateAudioParams</u>	Set audio device parameters for Alternate audio playback
<u>SetInputAudioParams</u>	Set audio device parameters for Main input
<u>ChangeModes</u>	Send a change mode request to the Apple device.
<u>SendHIDReport</u>	Send an input device command
<u>UpdateVehicleInformation</u>	Send vehicle information to the device
<u>SendConnect</u>	Send a connection request to a device
<u>GetIAPUrlWiFi</u>	Get the URL for a IAP connection via WiFi
<u>ReleaseIapUrl</u>	Release the URL for a IAP connection via WiFi
<u>Disconnect</u>	Disconnect the current session

Notes

The ICinemoCarPlay interface is in active development, closely following the development of CarPlay protocol by Apple. Future changes to this interface are likely.

5.17.1 Start

Start the CarPlay server.

Syntax

```
CinemoError Start();
```

Return value

If the method succeeds, it returns CinemoNoError. Otherwise it will return one of the following error codes:

CinemoErrorResources: If the acquisition of a resource has failed.

CinemoErrorSockInUse: If one of the configured host/port configurations is already in use.

CinemoErrorArguments: If one of the configured arguments is not valid or usable.

CinemoErrorDNSLookup: If the name lookup failed.

CinemoErrorNotImplemented: If a feature is requested that is not part of this Cinemo release.

Remarks

This method will start the CarPlay client threads, servers, and will begin advertising the CarPlay service using Bonjour mDNS protocol. The method returns immediately: nothing further will happen until the Apple device connects to the CarPlay client. This method is synchronous.

It is required to call [ICinemoCarPlay::SetInfo\(\)](#), [ICinemoCarPlay::SetCallbacks\(\)](#), [ICinemoCarPlay::ChangeModes\(\)](#) and [ICinemoCarPlay::RegisterScreen\(\)](#) before calling this method. It also advised to call [ICinemoCarPlay::RegisterHID\(\)](#), [ICinemoCarPlay::SetNightMode\(\)](#) and [ICinemoCarPlay::SetLimitedUI\(\)](#). If those methods are not called, Cinemo will use default values. The information provided by [ICinemoCarPlay::SetInfo\(\)](#) will be supplied to the Apple device when it connects and establishes a CarPlay session, the callback functions specified in [ICinemoCarPlay::SetCallbacks\(\)](#) are used to notify, amongst other events, when a session is established.

5.17.2 Stop

Stop the CarPlay service.

Syntax

```
CinemoError Stop();
```

Return value

This method returns CinemoNoError.

Remarks

This method will immediately terminate the CarPlay client threads, servers, and cease advertising the Bonjour mDNS service. After calling [Stop\(\)](#), it is possible to again call [Start\(\)](#), in which case a new session will be established if and when the Apple device reconnects. This method is synchronous.

Note that since this method deletes CarPlay objects, no other [ICinemoCarPlay](#) method may be in process or called during the execution of this method. The calling application is responsible for creating threads or other synchronization objects in order to avoid this, for example by waiting for applicable threads to finish before calling [Stop\(\)](#).

5.17.3 ForceKeyFrame

Force the Apple device to output a video keyframe.

Syntax

```
virtual CinemoError ForceKeyframe() = 0;
```

Return value

This method returns CinemoNoError.

Remarks

This method may be used to force the Apple device to output a video key frame. This method does not wait for the acknowledgement.

5.17.4 RequestSiri

Send a siri request to the apple device.

Syntax

```
virtual CinemoError RequestSiri(CINEMO_CARPLAY_SIRI_ACTION action) = 0;
```

Parameters

- **action**

The action that siri should take.

Return value

This method returns CinemoNoError.

Remarks

This method may be used to invoke siri with a specified action. Possible Actions are:

Name	Description
Prewarm	Indicate that the Apple device should begin preparing Siri. No audio or video resources will be taken
ButtonDown	Indicate to the Apple device that the Siri button has been pressed
ButtonUp	Indicate to the Apple device that the Siri button has been released

Those actions are represented by an enum.

```
typedef enum {
    CINEMO_CARPLAY_SIRI_ACTION_PREWARM = 1,
    CINEMO_CARPLAY_SIRI_ACTION_BUTTON_DOWN = 2,
    CINEMO_CARPLAY_SIRI_ACTION_BUTTON_UP = 3,
} CINEMO_CARPLAY_SIRI_ACTION;
```

This method does not wait for the acknowledgement.

5.17.5 SetLimitedUI

Enable or disable the limited UI.

Syntax

```
virtual CinemoError SetLimitedUI(CINEMO_CARPLAY_TRISTATE state) = 0;
```

Parameters

- **state**

Flag that indicates if the UI should be limited, or if this features is not available

Return value

This method returns CinemoNoError.

Remarks

This method may be used to indicate if the UI should be limited or not. The UI elements that can be limited are set with limited UI property. Possible values for this property are:

Value	Description
softKeyboard	Touch keyboard that appears on screen.
softPhoneKeypad	Touch phone keypad that appears on screen.
nonMusicLists	Lists of non-music items.
musicLists	Lists of music items.
japanMaps	Minor roads in Japan.

This method does not wait for the acknowledgement.

5.17.6 SetNightMode

Enable or disable the night mode.

Syntax

```
virtual CinemoError SetNightMode(CINEMO_CARPLAY_NIGHT_MODE mode) = 0;
```

Parameters

- **mode**

Flag that indicates if the night mode should be enabled.

Return value

This method returns CinemoNoError.

Remarks

This method may be used to indicate if it is dark outside and the night mode should be enabled. The support must be indicated by setting an initial value. Not initializing the night mode or setting it to CINEMO_CARPLAY_NIGHT_MODE_NOT_AVAILABLE will disable the night mode feature and the iPhone will determine if the night mode should be enabled or not. This method does not wait for the acknowledgement.

5.17.7 RequestUI

Request a specific UI to be shown.

Syntax

```
virtual CinemoError RequestUI(const char * szurl) = 0;
```

Parameters

- **szurl**

A URL for requesting a service. E.g maps: .

Return value

This method returns CinemoNoError.

Remarks

This method may be used to request a UI to be shown. This method does not wait for the acknowledgement.

5.17.8 SetCallbacks

Set callbacks for notification of client events.

Syntax

```
CinemoError SetCallbacks(
    [in] void * puser,
    [in] const CinemoCarPlaySessionCallbacks& callbacks);
```

Parameters

- **puser**

User-defined parameter passed to all callback functions.

- **callbacks**

User-defined callback functions.

Return value

This method returns CinemoNoError.

Remarks

The SetCallbacks() method is used to specify application-defined callbacks to receive notifications from a running CarPlay session. The CinemoCarPlayCallbacks structure is defined as follows:

```
typedef struct {
    CinemoCarPlaySessionInitialize fnInitialize;
    CinemoCarPlaySessionFinalize fnFinalize;
    CinemoCarPlaySessionStarted fnSessionStarted;
    CinemoCarPlaySessionModesChanged fnModesChanged;
    CinemoCarPlaySessionRequestUI fnRequestUI;
    CinemoCarPlaySessionDisableBluetooth fnDisableBluetooth;
    CinemoCarPlaySessionSetInputMode fnSetInputMode;
    CinemoCarPlaySessionGetAuthenticationCertificate fnGetCertificate;
    CinemoCarPlaySessionGetAuthenticationSignature fnGetSignature;
    CinemoCarPlaySessionAudioPrepare fnAudioPrepare;
    CinemoCarPlaySessionAudioStop fnAudioStop;
    CinemoCarPlaySessionAudioRampVolume fnAudioRampVolume;
    CinemoCarPlaySessionVideoPrepare fnVideoPrepare;
    CinemoCarPlaySessionVideoStop fnVideoStop;
    CinemoCarPlayHeartbeat fnHeartbeat;
    CinemoCarPlayServiceUpdate fnCarplayServiceUpdate;
} CinemoCarPlaySessionCallbacks;
```

NULL must be specified for callback functions which the application does not wish to receive. It is **strictly required** for calling code to initialize this structure to all zeros {0} before assigning the callback functions it wishes to receive, since Cinemo is likely to extend this structure with additional callbacks in future releases.

The following fields are defined:

- **fnInitialize**

Called when a session is initialized.

- **fnFinalize**

Called when a session is finalized.

- **fnSessionStarted**

Called when a session is started. From now on messages can be send to the iPhone

- **fnModesChanged**

Called when a mode change is signaled by the Apple device.

- **fnRequestUI**

Called when Apple device requests accessory UI.

- **fnDisableBluetooth**

Called when Apple device requests to disable Bluetooth connections.

- **fnSetInputMode**

Called to request a specific input device.

- **fnGetCertificate**

Called when Apple device requests an authentication certificate.

- **fnGetSignature**

Called when Apple device requests an authentication challenge response.

- **fnAudioPrepare**

Called when preparing an audio stream.

- **fnAudioStop**

Called when an audio stream is stopped.

- **fnAudioRampVolume**

Called when Apple device requests to duck or unduck an audio stream.

- **fnVideoPrepare**

Called when preparing the video screen.

- **fnVideoStop**

Called when video screen is stopped.

- **fnHeartbeat**

Called on a regular interval, to show that the CarPlay client is still alive.

Please see ICinemoCarPlay Callbacks for further details on each callback method. This method is synchronous.

5.17.9 SetInfo

Set client information.

Syntax

```
CinemoError SetInfo(
    [in] struct ICinemoMetapool * ppool);
```

Parameters

- **ppool**

ICinemoMetapool containing CarPlay client information.

Return value

This method returns CinemoNoError.

Remarks

The information specified here will be delivered to the Apple device when it requests accessory information during establishment of a CarPlay session. Cinemo currently recognizes the following metadata:

CINEMO_CARPLAY	TYPE
CINEMO_CARPLAY_FIRMWARE_REVISION	UTF-8
CINEMO_CARPLAY_HARDWARE_REVISION	UTF-8
CINEMO_CARPLAY_BLUETOOTH_ID	UTF-8 multiple index possible
CINEMO_CARPLAY_HID_LANGUAGES	UTF-8 multiple index possible
CINEMO_CARPLAY_LIMITED_UI_ELEMENT	UTF-8 multiple index possible
CINEMO_CARPLAY_MANUFACTURER	UTF-8
CINEMO_CARPLAY_OEM_ICON_VISIBLE	UINT32
CINEMO_CARPLAY_OEM_ICONS	UTF-8
CINEMO_CARPLAY_OEM_ICON_LABEL	UTF-8
CINEMO_CARPLAY_OEM_ICON	Image
CINEMO_CARPLAY_RIGHT_HAND_DRIVE	UINT32
CINEMO_CARPLAY_AUDIO_MAIN_FORMATS	UTF-8
CINEMO_CARPLAY_AUDIO_MAIN_HIGH_FORMATS	UTF-8
CINEMO_CARPLAY_AUDIO_ALT_FORMATS	UTF-8
CINEMO_CARPLAY_DEVICE_ID	UTF-8
CINEMO_CARPLAY_OS_INFO	UTF-8
CINEMO_CARPLAY_MODEL	UTF-8
CINEMO_CARPLAY_AUDIO_LATENCIES	UTF-8

CINEMO_CARPLAY	TYPE
CINEMO_CARPLAY_HEARTBEAT_TIMEOUT	UINT32
CINEMO_CARPLAY_EXTENDED_FEATURES	UTF-8

The above list is not final, and will be updated in future versions of this document. It is worth noting that since the information requested by an Apple device is in the form of key-value pairs, which corresponds exactly to the data representation of ICinemoMetapool, it is possible to add additional information to ICinemoMetapool which may be understood directly by the Apple device, even if Cinemo defines no CINEMO_CARPLAY for a specific key.

For example, the following code snippet is valid:

```
void SetOEMIconLabel(ICinemoMetapool * p, const char * szLabel) {
    p->AddItem(
        "oemIconLabel",
        CINEMO_METATYPE_UTF8,
        CINEMO_METALANG_UNSPECIFIED,
        0,
        0,
        szLabel,
        strlen(szLabel));
}
```

This code works because CINEMO_CARPLAY_OEM_ICON_LABEL is internally defined by Cinemo as the string "oemIconLabel", which is directly understood by the Apple device. This method must be called before the start of Cinemo CarPlay. If it is not called all subsequent calls to [ICinemoCarPlay::Start\(\)](#) will fail. This method is synchronous. The initial modes are not part of meta pool and can be set by calling [ICinemoCarPlay::ChangeModes\(\)](#).

5.17.10 SetMainAudioParams

5.17.11 SetAlternateAudioParams

5.17.12 SetInputAudioParams

Set audio device parameters.

Syntax

```
CinemoError SetMainAudioParams(  
    [in] const CinemoAudioParams& params);  
CinemoError SetAlternateAudioParams(  
    [in] const CinemoAudioParams& params);  
CinemoError SetInputAudioParams(  
    [in] const CinemoAudioParams& params);
```

Parameters

- **params**

Audio device parameters.

Return value

This method returns CinemoNoError.

Remarks

These methods changes the parameters for platform-specific audio input and output devices, and are functionally equivalent to ICinemoPlayer::SetAudioParams(). The only point of distinction is that ICinemoCarPlay exposes three methods, for independent configuration of main audio, alternate audio and microphone input.

These methods may be called at any time. If they are called before playback/recording has begun, e.g. during AudioPrepare(), the parameters will be saved and become effective when the corresponding playback or recording device is next instantiated, e.g. immediately after the application returns from the AudioPrepare() callback. If they are called during playback or recording, the parameters will be stored and used for the next session. The currently active session is not affected.

Note that both Main and Alternative audio output devices may be active simultaneously, but only one audio input device is currently supported by Apple (Main input). This method is synchronous.

5.17.13 ChangeModes

Send a change mode request to the Apple device.

Syntax

```
CinemoError ChangeModes(
    [in] const CinemoCarPlayModeChanges& mode_changes,
    [in] const char * szreason);
```

Parameters

- **mode_changes**

Describe mode changes required by the accessory.

- **szreason**

Optional UTF-8 zero terminated reason for the change mode request, may be NULL.

Return value

If the method succeeds, it returns CinemoNoError. Otherwise it will return one of the following error codes:

CinemoErrorResources: If the acquisition of a resource has failed.

Remarks

This method allows an accessory to change modes, that is, to inform the Apple device which screen and audio resources it currently requires, and the relative priority of these requirements. The mode change information is passed in the CinemoCarPlayModeChanges structure:

```
typedef struct {
    CinemoCarPlayResourceChange screen;           /* request for the screen */
    CinemoCarPlayResourceChange main_audio;        /* request for main audio */
    CINEMO_CARPLAY_SPEECHMODE speech;            /* request for speech */
    CINEMO_CARPLAY_TRISTATE phone_call;          /* request for phone call */
    CINEMO_CARPLAY_TRISTATE turn_by_turn;         /* request for navigation */
} CinemoCarPlayModeChanges;
```

The ChangeModes() method sends the mode change request to the Apple device. The Apple device will then either accept the request and determine a new mode, or reject the request because it is not compatible with current resource constraints (in which case a CinemoErrorResources code will be returned). If the method succeeds, the accessory will be informed of the new mode, as determined by the Apple device, in the ModesChanged() callback notification. A return of CinemoNoError does not mean that the device will use the mode requested. The exact timing of this callback notification is not currently specified, and may be delayed. If this method is called before a call to [ICinemoCarPlay::Start\(\)](#) this information is stored and send later to the apple device when requested. This method is asynchronous.

5.17.14 SendHIDReport

Send an input device command.

Syntax

```
CinemoError SendHIDReport(  
    [in] uint32 index,  
    [in] const void * preport,  
    [in] uint32 nbytes);
```

Parameters

- **index**

HID descriptor index.

- **preport**

USB-style HID event report.

- **nbytes**

Length of the HID event report in bytes.

Return value

If the method succeeds, it returns CinemoNoError. Otherwise it will return one of the following error codes:

CinemoErrorArguments: If the name of the device could not be found.

Remarks

This method will forward the HID report data for a previously registered HID with the same name to the apple device. The SendHIDReport() method is effectively a pass-thru, allowing an application full and complete control of user input to the CarPlay session. This method does not wait for the acknowledgement.

5.17.15 UpdateVehicleInformation

Send vehicle information to the device

Syntax

```
CinemoError UpdateVehicleInformation(  
    [in] const CinemoCarPlayVehicleInformation_t& info           /* the vehicle information */
```

Parameters

- **info**

The vehicle information

Return value

If the method succeeds, it returns CinemoNoError. Otherwise it will return one of the following error codes:

CinemoErrorResources: Internal resources could not be allocated/deallocated.

Remarks

This method will send vehicle information to a apple device. This method is asynchronous.

5.17.16 SendConnect

Send a connection request to a device

Syntax

```
CinemoError SendConnect(  
    [in] const char* device_id,  
    [in] const char* service_name)
```

Parameters

- **device_id**

Device from the control client for this service

- **service_name**

The name of the service

Return value

If the method succeeds, it returns Cinemo.NoError. Otherwise it will return one of the following error codes:

CinemoErrorResources: Internal resources could not be allocated/deallocated.

CinemoErrorArguments: The requested service or host could not be resolved on the given interface index

Remarks

This method will send a request to a apple device to connect a new CarPlay session.

5.17.17 GetIAPUrlWifi

Get the URL for a IAP connection via WiFi

Syntax

```
CinemoError GetIAPUrlWifi(  
    [out] struct ICinemoUTF8** url); /* the url that can be used to connect  
    to IAP */
```

Parameters

- **url**

The URL to connect to the IAP

Return value

If the method succeeds, it returns CinemoNoError. Otherwise it will return one of the following error codes:

CinemoErrorResources: Internal resources could not be allocated/deallocated.

CinemoErrorArguments: There is no active session that could do an IAP connection.

Remarks

This method will setup the internal transport for the cinemo IAP library. This method is synchronous.

5.17.18 **ReleaseIapUrl**

Release the URL for a IAP connection via WiFi

Syntax

```
CinemoError ReleaseIapUrl();
```

Return value

If the method succeeds, it returns CinemoNoError.

Remarks

This method will release the internal transport for the cinemo IAP library. This method is synchronous.

5.17.19 Disconnect

Disconnect the current active session

Syntax

```
CinemoError Disconnect(  
    [in] const char* device_id,  
    [in] const char* service_name,  
    [in] const char* ip_address  
);
```

Parameters

- **device_id**

The device given by the ServiceUpdate callback

- **service_name**

The service name given by the ServiceUpdate callback

- **ip_address**

The IP address of the iPhone or null

Return value

If the method succeeds, it returns Cinemo.NoError. Otherwise it will return one of the following error codes:

CinemoErrorResources: Internal resources could not be allocated/deallocated.

CinemoErrorArguments: There is no active session.

Remarks

This method will either terminate the current active session if the ip address is not set or it will check if the currently active session is connected to the provided ip address and only if they match the current session will be terminated. This method is synchronous.

5.18 ICinemoCarPlay Callbacks

Applications using ICinemoCarPlay interface become aware of CarPlay state changes and events by means of callback functions. All callbacks which an application wishes to receive should be specified in the SetCallbacks() method before calling Start().

If an application does not wish to implement a particular callback function, it can pass NULL for that function, in which case Cinemo supplies a default implementation.

ICinemoCarPlay Callbacks

CALLBACK	DESCRIPTION
<u>Initialize</u>	A CarPlay session has started.
<u>Finalize</u>	A CarPlay session has finished
<u>Started</u>	A CarPlay session has been started
<u>ModesChanged</u>	Apple device has changed modes
<u>RequestUI</u>	Apple device request to show accessory UI
<u>DisableBluetooth</u>	Apple device request to disable Bluetooth connections
<u>SetInputModule</u>	Apple device request to use a specific input device
<u>GetAuthenticationCertificate</u>	Apple device requests an authentication certificate
<u>GetAuthenticationSignature</u>	Apple device requests an authentication challenge response
<u>AudioPrepare</u>	Prepare an audio stream for playback or input
<u>AudioStop</u>	An audio stream has stopped
<u>AudioRampVolume</u>	Apple device requests to duck or unduck main audio
<u>VideoPrepare</u>	Prepare the video screen for playback
<u>VideoStop</u>	The video screen has stopped
<u>Heartbeat</u>	The car play is active
<u>ServiceUpdate</u>	A new CarPlay service has been found
<u>ConnectionResult</u>	The result of a connection attempt.

Notes

An application should make no assumptions about the order in which callback functions are executed, or on which thread they are executed. Cinemo's CarPlay implementation runs multiple internal threads, all of which may deliver asynchronous events or notifications. An application must ensure its callback functions are capable of executing concurrently on multiple threads (for example, by implementing its own mutex or other queueing mechanism).

5.18.1 Initialize

A CarPlay session has started.

Syntax

```
CinemoError CinemoCarPlaySessionInitialize(  
    [in] void * puser,  
    [in] const char * szname,  
    [in] const char * uuid,  
    [in] const char * model);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **szname**

The user-friendly name of the Apple device which established this session, for example, "Cinemo's iPhone 5S".

- **uuid**

The device id sent by the Apple device

- **model**

The model name of the Apple device

Return value

If this method is successful it should return CinemoNoError. Every other error code will stop the session initialization.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the CarPlay session has been established by the Apple device. If the application does NOT wish a session to be established, it should return an error code other than CinemoNoError.

The default Cinemo implementation of this function does nothing.

5.18.2 Finalize

A CarPlay session has finished.

Syntax

```
CinemoError CinemoCarPlaySessionFinalize(  
    [in] void * puser);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the CarPlay session has finished, for example, if either the accessory or the Apple device terminated the session. The return value from this function is ignored.

The default Cinemo implementation of this function does nothing.

5.18.3 Started

A CarPlay session has been started.

Syntax

```
CinemoError CinemoCarPlaySessionStarted(  
    [in] void * puser);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the CarPlay session has been started. This state information is important because it is not valid to send any commands or messages to the iPhone until after this method has been called.

The default Cinemo implementation of this function does nothing.

5.18.4 ModesChanged

Apple device has changed modes.

Syntax

```
CinemoError CinemoCarPlaySessionModesChanged(
    [in] void * puser,
    [in] const CinemoCarPlayCurrentMode& mode);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **mode**

New mode specified by the Apple device.

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the Apple device send a “ModesChanged” notification to update the current mode.

```
typedef struct {
    CINEMO_CARPLAY_ENTITY screen;           /* owner of the screen */
    CINEMO_CARPLAY_ENTITY main_audio;        /* owner of main audio */
    CINEMO_CARPLAY_ENTITY phone_call;        /* owner of phone call */
    CINEMO_CARPLAY_ENTITY turn_by_turn;       /* owner of navigation */
    CINEMO_CARPLAY_ENTITY speech;            /* owner of speech */
    CINEMO_CARPLAY_SPEECHMODE speechmode;    /* the current speech mode */
} CinemoCarPlayCurrentMode;
```

Mode changes originating from the Apple device are communicated directly to the accessory as an update to the current mode, and are not preceded by a mode change request. The accessory must honor any updates to the current mode with 100ms of receiving them from the Apple device, for example, it must relinquish control of the screen within 100ms if screen ownership is transferred to the Apple device.

Since the accessory has no choice but to accept the current mode, the return value from this function is ignored. The default Cinemo implementation of this function does nothing.

5.18.5 RequestUI

Apple device request to show accessory UI.

Syntax

```
CinemoError CinemoCarPlaySessionRequestUI(  
    [in] void * puser  
    [in] const char* url);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **url**

A url not specified by apple yet

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the Apple device requests for the accessory's UI to be shown.

As currently understood by Cinemo: the home page of Apple's CarPlay screen can be configured with an application defined icon and text label. The "RequestUI" message is sent to the accessory when the user presses this button. See SetInfo() method for details about configuring the icon.

The return value from this function is ignored. The default Cinemo implementation of this function does nothing.

5.18.6 DisableBluetooth

Apple device request to disable Bluetooth connections.

Syntax

```
CinemoError CinemoCarPlaySessionDisableBluetooth(  
    [in] void * puser,  
    [in] const char * szdevice);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **szdevice**

MAC address of the Bluetooth device to disable.

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the Apple device requests for a Bluetooth device connection to be disabled.

At a minimum, on receipt of this command, the accessory must immediately disable any existing connections with the specified Apple device for the following Bluetooth profiles:

- Hands Free Profile
- Advanced Audio Distribution Profile
- A/V Remote Control Profile

In addition, the accessory must not attempt to connect any of the above-listed Bluetooth profiles with the same Apple device while the two are already connected, i.e., while a CarPlay session is in progress.

The return value from this function is ignored. The default Cinemo implementation of this function does nothing.

5.18.7 SetInputMode

Apple device request to use a specific input device.

Syntax

```
CinemoError CinemoCarPlaySessionSetInputMode (
    [in] void * puser,
    [in] const char* uuid);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **uuid**

The UUID of the input device to use.

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the Apple device request a specific input device.

The return value from this function is ignored. The default Cinemo implementation of this function does nothing.

5.18.8 GetAuthenticationCertificate

Apple device requests an authentication certificate.

Syntax

```
CinemoError CinemoCarPlaySessionGetAuthenticationCertificate(
    [in] void * puser,
    [out] struct ICinemoBlob ** pp);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **pp**

Return an ICinemoBlob interface containing the requested certificate data.

Return value

If this method is successful it should return CinemoNoError. If the certificate cannot be retrieved it is possible to return CinemoErrorNotImplemented to use the cinemo internal method. Every other error code will stop the authentication process.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the Apple device requests an X.509 authentication certificate. Digital iPod Out is an authenticated solution, requiring the use of an MFi authentication coprocessor (2.0B or later) obtained through Apple. Apple devices will only stream to authenticated accessories.

The default Cinemo implementation of this function will return the certificate provided by supported i2c hardware. This function need only be overridden if an application wishes to provide the certificate itself, for example, if it requires exclusive access to the the authentication coprocessor, or if the X.509 certificate is already cached by the system.

The ICinemoBlob interface will be automatically released when it is no longer required.

5.18.9 GetAuthenticationSignature

Apple device requests an authentication challenge response.

Syntax

```
CinemoError CinemoCarPlaySessionGetAuthenticationSignature(
    [in] void * puser,
    [in] const void * pchallenge,
    [in] int nbytes,
    [out] struct ICinemoBlob ** pp);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **pchallenge**

Challenge data supplied by the Apple device.

- **nbytes**

Challenge data size in bytes.

- **pp**

Return an ICinemoBlob interface containing the requested response data.

Return value

If this method is successful it should return CinemoNoError. If the signature cannot be retrieved it is possible to return CinemoErrorNotImplemented to use the cinemo internal method. Every other error code will stop the authentication process.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the Apple device requests an authentication challenge response.

The default Cinemo implementation of this function will return the response provided by supported i2c hardware. This function need only be overridden if an application wishes to provide the response itself, for example, if it requires exclusive access to the the authentication coprocessor.

The ICinemoBlob interface will be automatically released when it is no longer required.

5.18.10 AudioPrepare

Prepare an audio stream for playback or input.

Syntax

```
CinemoError CinemoCarPlaySessionAudioPrepare (
    [in] void * puser,
    [in] const CinemoCarPlayAudio& audio);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **audio**

Properties of the audio stream.

Return value

If this method is successful it should return CinemoNoError. Every other error code will stop the audio stream setup.

Remarks

If specified in SetCallbacks(), this application defined function will be called when an audio stream is prepared for playback or microphone input. Note that an audio stream, as specified by Apple, may be full-duplex or half-duplex. That is, it may be used for both playback and microphone input, or for playback only. There are no input-only streams.

An application may use this callback to setup audio hardware appropriately or change audio device parameters dynamically with [ICinemoCarPlay::SetMainAudioParams\(\)](#), [ICinemoCarPlay::SetAlternateAudioParams\(\)](#), [ICinemoCarPlay::SetInputAudioParams\(\)](#) calls. The stream properties are defined by the CinemoCarPlayAudio structure:

```
typedef struct {
    CinemoAudioFormat format;
    const char * type;
    const char * streamType;
    uint32 latencyMs;
    uint32 input;
    uint32 varispeed;
} CinemoCarPlayAudio;
```

The following fields are defined:

- **format**

The format of the audio stream, including sample rate, channels, bit depth, etc.

- **type**

The audio type, defined by Apple as one of the following string values: default, media, measurement, telephony, speechRecognition, spokenAudio or alert.

- **streamType**

The audio stream type, defined by Apple as one of the following string values: GeneralAudio, MainAudio, AltAudio or Screen.

- **latencyMs**

The latency of the audio stream, in milliseconds.

- **input**

Boolean flag – if set to "0", the stream is half-duplex, for playback only. Otherwise, the stream is full-duplex, for both playback and microphone input.

- **varispeed**

Boolean flag indicating support for fine-grained skew compensation.

5.18.11 AudioStop

An audio stream has stopped.

Syntax

```
CinemoError CinemoCarPlaySessionAudioStop(  
    [in] void * puser,  
    [in] const CinemoCarPlayAudio& audio);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **audio**

Properties of the audio stream.

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when an audio stream has been stopped. The CinemoCarPlayAudio properties passed to this method are identical to those provided in a previous call to the AudioPrepare() callback.

An application may use this callback to free hardware audio resources. The return value from this function is ignored.

5.18.12 AudioRampVolume

Apple device requests to duck or unduck main audio.

Syntax

```
CinemoError CinemoCarPlaySessionAudioRampVolume (
    [in] void * puser,
    [in] double volume,
    [in] uint32 durationMs);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **volume**

Suggested final attenuation of the audio in units from 0 to 1 (linear percentage).

- **durationMs**

Number of milliseconds the ramp should last.

Return value

If this method is successful it should return CinemoNoError. If for some reason the request cannot be fulfilled the method can return CinemoErrorNotImplemented to indicate that the internal implementation should be used.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the Apple device requests to ramp main audio volume up or down, e. g. before and after a turn-by-turn announcement from the Apple device.

Cinemo provides a compliant internal implementation of duck and unduck of the Apple device main audio stream. An application may wish to override Cinemo's default behaviour if, for example, the application has access to system audio hardware which provides a hardware or otherwise superior implementation. In any case, an implementation of this function should be provided to support compliant ducking and unducking of accessory main audio.

The default Cinemo implementation of this function will duck and unduck audio streams according to Apple's specifications.

The volume parameter $v_{lin\%}$ is calculated from the original attenuation v_{db} :

$$v_{lin\%} = \begin{cases} 0 & \text{if } v_{db} \leq -144 \\ 10^{\frac{v_{db}}{20}} & \text{if } -144 < v_{db} < 0 \\ 1 & \text{if } v_{db} \geq 0 \end{cases}$$

5.18.13 VideoPrepare

Prepare the video screen for playback.

Syntax

```
CinemoError CinemoCarPlaySessionVideoPrepare (
    [in] void * puser,
    [in] const CinemoCarPlayVideo& video);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **video**

Properties of the video stream.

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the video screen is prepared for playback. An application may use this callback to setup video decoding and rendering hardware appropriately. This callback must block until an associated [ICinemoPlayer](#) instance has entered domain CINEMO_DOMAIN_PLAY and current CinemoVideoParams setting is not in disabled state.

If the associated [ICinemoPlayer](#) instance cannot enter domain CINEMO_DOMAIN_PLAY (indicated by CINEMO_EC_ERROR or CINEMO_EC_OPEN with an error code other than CinemoNoError) or before the CarPlay instance is to be stopped or finally released it needs be made sure that this callback is not blocked. The return value from this function is ignored.

5.18.14 VideoStop

The video screen has stopped.

Syntax

```
CinemoError CinemoCarPlaySessionVideoStop(  
    [in] void * puser,  
    [in] const CinemoCarPlayVideo& video);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **video**

Properties of the video stream.

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called when the video stream has been stopped. The CinemoCarPlayVideo properties passed to this method are identical to those provided in a previous call to the VideoPrepare() callback.

An application may use this callback to free hardware video resources. The return value from this function is ignored.

5.18.15 Heartbeat

The car play is active and the thread is not blocked or deadlocked.

Syntax

```
CinemoError CinemoCarPlayHeartbeat(  
    [in] void * puser);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called on a regular basis. The interval can be configured via the meta pool. The return value from this function is ignored.

5.18.16 ServiceUpdate

This method informs the host that a device offering a new CarPlay session was found.

Syntax

```
CinemoError CinemoCarPlayServiceUpdate(
    [in] void* puser,
    [in] CinemoCarPlayServiceType type,
    [in] const char* device_id,
    [in] const char* service_name);
```

Parameters

- **puser**

User-defined parameter specified in SetCallbacks().

- **type**

The type of the update.

- **device_id**

The device_id of the service from the control client.

- **service_name**

The name of the service.

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called whenever a new service is discovered. The return value from this function is ignored.

5.18.17 ConnectionResult

Please note that this API has been deprecated and might no longer be available in upcoming releases of the Cinemo SDK.

The result of a connection attempt.

Syntax

```
CinemoError CinemoCarPlayConnectionResult(  
    [in] void* puser,  
    [in] CinemoError error);
```

Parameters

- **puser**
User-defined parameter specified in SetCallbacks().
- **error**
The result of the last connection attempt.

Return value

If this method is successful it should return CinemoNoError.

Remarks

If specified in SetCallbacks(), this application defined function will be called to report the result of a connection attempt. The return value from this function is ignored.

6. Cinemo Transport Library

The Cinemo Transport Library interface is defined in a C header file which is provided by your Cinemo representative.

6.1 General Remarks

Cinemo provides an interface for using external transport libraries to communicate with third party devices, e.g. Apple or Bluetooth devices. In this document the interface is described and important correlations of the individual parts are explained.

6.1.1 Requirements

A custom implementation of the Cinemo transport library has to fulfill the following requirements:

- It must be binary compatible with regular Cinemo SDK releases for the target platform.
- It must not throw any exceptions.

If these requirements are met, Cinemo will be able to use the custom transport library implementation for communication with an external device.

If the transport library is provided to Cinemo, either as binary or as source code, it is possible to include it in a release of the Cinemo SDK.

6.2 Transport Interface

Transport Interface Methods

METHOD	DESCRIPTION
<u>Create_Transport_Custom</u>	Create function
<u>protocol_version</u>	Protocol version of this implementation
<u>ctli_handle</u>	Custom transport handle
<u>ctli_delete_transport_fn</u>	Delete the transport instance
<u>ctli_receive_fn</u>	Receive data
<u>ctli_send_fn</u>	Send data
<u>ctli_abort_fn</u>	Abort all blocking operations
<u>ctli_get_param_fn</u>	Parameter request from Cinemo
<u>ctli_set_param_fn</u>	A parameter changed
<u>ctli_audio_create_fn</u>	Create audio streaming object
<u>ctli_audio_delete_fn</u>	Delete audio streaming object
<u>ctli_audio_receive_fn</u>	Receive audio data
<u>ctli_audio_abort_fn</u>	Abort blocking audio operations
<u>ctli_audio_get_params_fn</u>	Audio parameter request from Cinemo
<u>ctli_audio_set_params_fn</u>	New audio parameters are available
<u>ctli_dataport_select_fn</u>	Select the specified dataports
<u>ctli_dataport_send_fn</u>	Send data via dataport
<u>ctli_dataport_recv_fn</u>	Receive data via dataport
<u>ctli_dataport_cancel_fn</u>	Cancel dataport
<u>ctli_dataport_enable_fn</u>	Enable dataport
<u>ctli_dataport_get_param_fn</u>	Dataport parameter request from Cinemo
<u>ctli_dataport_set_param_fn</u>	Dataport parameter change request from Cinemo

The interface for the Cinemo transport library is provided in the form of a header file.

```
*****
Project:      Cinemo Media Engine

Developed by:   Cinemo GmbH
                Karlsruhe, Germany

Copyright (c) 2009-2018 Cinemo GmbH and its licensors.
All rights reserved.

This software is supplied only under the terms of a license agreement,
nondisclosure agreement or other written agreement with Cinemo GmbH.
Use, redistribution or other disclosure of any parts of this software
```

is prohibited except in accordance with the terms of such written agreement with Cinemo GmbH. This software is confidential and proprietary information of Cinemo GmbH and its licensors.

In case of licensing questions please contact: sales@cinemo.com

```
*****
#ifndef CINEMO_TRANSPORT_H_INCLUDED
#define CINEMO_TRANSPORT_H_INCLUDED

#define CINEMO_TRANSPORT_LIBRARY_PROTOCOL_VERSION 5

#define CTLI_NO_ERROR 0
#define CTLI_ERROR_ARGUMENTS -1
#define CTLI_ERROR_AUDIOPARAMS -2
#define CTLI_ERROR_CANCEL -3
#define CTLI_ERROR_DECODE -4
#define CTLI_ERROR_IO -5
#define CTLI_ERROR_NOTCONNECTED -6
#define CTLI_ERROR_NOTENOUGHSPACE -7
#define CTLI_ERROR_NOTIMPLEMENTED -8
#define CTLI_ERROR_OPEN -9
#define CTLI_ERROR_OVERFLOW -10
#define CTLI_ERROR_RESOURCES -11
#define CTLI_ERROR_AUDIOSETUP -12
#define CTLI_ERROR_CONFIGURATION -13
#define CTLI_ERROR_IDLE -14
#define CTLI_ERROR_UNAVAILABLE -15
#define CTLI_ERROR_DRIVER_FAILURE -16

typedef enum CTLI_PARAM_PROTOCOL_VALUES {
    CTLI_PARAM_PROTOCOL_UNKNOWN = 0, /* unknown protocol */
    CTLI_PARAM_PROTOCOL_USB, /* USB */
    CTLI_PARAM_PROTOCOL_BLUETOOTH, /* bluetooth */
    CTLI_PARAM_PROTOCOL_WIFI, /* wifi */
} CTLI_PARAM_PROTOCOL_VALUES;

typedef enum CTLI_PARAM_STATE_VALUES {
    CTLI_PARAM_STATE_UNKNOWN = 0, /* unknown state */
    CTLI_PARAM_STATE_DEVICEMODE, /* device is connected in device mode */
    CTLI_PARAM_STATE_HOSTMODE, /* device is connected in host mode */
    CTLI_PARAM_STATE_DISCONNECTED, /* device has been disconnected */
} CTLI_PARAM_STATE_VALUES;

// for non-USB transports CTLI_PARAM_STATE_DEVICEMODE should
// be used if the connection is established

typedef enum CTLI_PARAM {
    CTLI_PARAM_PROTOCOL = 0,
    CTLI_PARAM_STATE,
    CTLI_PARAM_USB_VID,
    CTLI_PARAM_USB_PID,
```

```

CTLI_PARAM_USB_MANUFACTURER,
CTLI_PARAM_USB_PRODUCT,
CTLI_PARAM_USB_SERIALNO,
CTLI_PARAM_NCM_IF_NUM,
CTLI_PARAM_BLUETOOTH_MAC_ADDRESS,
CTLI_PARAM_BLUETOOTH_ADAPTER_NAME,
CTLI_PARAM_BLUETOOTH_DEVICE_NAME,           /* char[249] max 248 bytes, should be encoded
                                             in UTF-8 */
CTLI_PARAM_BLUETOOTH_AVRCP_VERSION,          /* unsigned short; two bytes integer defined
                                             AVRCP specification */
CTLI_PARAM_BLUETOOTH_AVRCP_FEATURESFLAGS,    /* unsigned short; two bytes integer defined
                                             AVRCP specification */
CTLI_PARAM_BLUETOOTH_AVRCP_BROWSING_PSM,     /* unsigned short; two bytes integer defined
                                             AVRCP specification */
CTLI_PARAM_BLUETOOTH_L2CAP_IMTU,              /* unsigned short; MTU for receiving data from
                                             device */
CTLI_PARAM_BLUETOOTH_L2CAP_OMTU               /* unsigned short; MTU for sending data to
                                             device */
} CTLI_PARAM;

union cctl_params {
    CTLI_PARAM_PROTOCOL_VALUES protocol;
    CTLI_PARAM_STATE_VALUES state;
    unsigned short usb_vid;
    unsigned short usb_pid;
    char usb_manufacturer[255];
    char usb_product[255];
    char usb_serialno[255];
    unsigned char ncm_if_num;
    char bt_mac_address[6];
    char bt_adapter_name[255];
    char bt_device_name[249]; // UTF-8 max 248 bytes
    unsigned short bt_avrcp_version;
    unsigned short bt_avrcp_fflags;
    unsigned short bt_avrcp_browsing_psm;
    unsigned short bt_l2cap_imtu;
    unsigned short bt_l2cap_omtu;
};

typedef enum CTLI_AUDIO_CONTAINER {
    CTLI_AUDIO_CONTAINER_UNSPECIFIED = 0,
    CTLI_AUDIO_CONTAINER_RAW,
    CTLI_AUDIO_CONTAINER_LATM,
    CTLI_AUDIO_CONTAINER_AVDTP
} CTLI_AUDIO_CONTAINER;

typedef enum CTLI_AUDIO_CODEC {
    CTLI_AUDIO_CODEC_UNSPECIFIED = 0,
    CTLI_AUDIO_CODEC_PCM,
    CTLI_AUDIO_CODEC_MP3,
    CTLI_AUDIO_CODEC_AAC,
    CTLI_AUDIO_CODEC_SBC,
}

```

```

    CTLI_AUDIO_CODEC_APTX
} CTLI_AUDIO_CODEC;

typedef enum CTLI_AUDIO_SAMPLETYPE {
    CTLI_AUDIO_SAMPLETYPE_UNSPECIFIED = 0,
    CTLI_AUDIO_SAMPLETYPE_S16LE,
    CTLI_AUDIO_SAMPLETYPE_S24LE,
} CTLI_AUDIO_SAMPLETYPE;

typedef struct ctli_audio_params {
    CTLI_AUDIO_CONTAINER container;
    CTLI_AUDIO_CODEC codec;
    CTLI_AUDIO_SAMPLETYPE samplertype;
    int samplerate;
    int channels;
    int bits;
} ctli_audio_params;

typedef enum CTLI_DATAPORT_TRANSITION {
    CTLI_DATAPOINT_UNKNOWN = 0,
    CTLI_DATAPOINT_OFFLINE_TO_ONLINE,
    CTLI_DATAPOINT_ONLINE_TO_OFFLINE,
} CTLI_DATAPORT_TRANSITION;

typedef struct ctli_dataport_state {
    unsigned short id;
    unsigned char state_changed;
    CTLI_DATAPORT_TRANSITION transition;
} ctli_dataport_state;

typedef enum CTLI_DATAPORT_PARAM {
    CTLI_DATAPORT_PARAM_STATE = 0,
} CTLI_DATAPORT_PARAM;

typedef enum CTLI_DATAPORT_STATE {
    CTLI_DATAPORT_STATE_OFFLINE = 0,
    CTLI_DATAPORT_STATE_ONLINE,
} CTLI_DATAPORT_STATE;

union ctli_dataport_params {
    CTLI_DATAPORT_STATE state;
};

typedef struct ctli_hid_report_descriptor {
    unsigned short length;
    const void* pdata;
} ctli_hid_report_descriptor;

typedef struct ctli_create_params {
    const char* szurl;
    const char* szoption;
    const char* szdataports;
}

```

```

    ctli_hid_report_descriptor* phid;
    unsigned int hid_count;
} ctli_create_params;

typedef void* ctli_handle;

typedef int (*ctli_delete_transport_fn)(ctli_handle tphandle);
typedef int (*ctli_receive_fn)(ctli_handle tphandle, void* pdest, unsigned int npdest);
typedef int (*ctli_send_fn)(ctli_handle tphandle, const void* psrc, unsigned int npsrc);
typedef int (*ctli_abort_fn)(ctli_handle tphandle);
typedef int (*ctli_get_param_fn)(ctli_handle tphandle, CTLI_PARAM param_type, ctli_params*
    params);
typedef int (*ctli_set_param_fn)(ctli_handle tphandle, CTLI_PARAM param_type, const
    ctli_params* params);
typedef int (*ctli_audio_create_fn)(ctli_handle tphandle);
typedef int (*ctli_audio_delete_fn)(ctli_handle tphandle);
typedef int (*ctli_audio_receive_fn)(ctli_handle tphandle, void* pdest, unsigned int npdest);
typedef int (*ctli_audio_abort_fn)(ctli_handle tphandle);
typedef int (*ctli_audio_get_params_fn)(ctli_handle tphandle, ctli_audio_params* params);
typedef int (*ctli_audio_set_params_fn)(ctli_handle tphandle, const ctli_audio_params* params)
    ;
typedef int (*ctli_dataport_select_fn)(ctli_handle tphandle, ctli_dataport_state* state,
    unsigned int num_entries);
typedef int (*ctli_dataport_send_fn)(ctli_handle tphandle, unsigned short id, const void* psrc
    , unsigned int npsrc);
typedef int (*ctli_dataport_recv_fn)(ctli_handle tphandle, unsigned short id, void* pdest,
    unsigned int npdest);
typedef int (*ctli_dataport_cancel_fn)(ctli_handle tphandle, unsigned short id);
typedef int (*ctli_dataport_enable_fn)(ctli_handle tphandle, unsigned short id);
typedef int (*ctli_dataport_get_param_fn)(ctli_handle tphandle, unsigned short id,
    CTLI_DATAPORT_PARAM param_type, ctli_dataport_params* params);
typedef int (*ctli_dataport_set_param_fn)(ctli_handle tphandle, unsigned short id,
    CTLI_DATAPORT_PARAM param_type, const ctli_dataport_params* params);

typedef struct ctli_context {
    unsigned short protocol_version;
    ctli_handle tphandle;
    ctli_delete_transport_fn delete_transport;
    ctli_receive_fn receive;
    ctli_send_fn send;
    ctli_abort_fn abort;
    ctli_get_param_fn get_param;
    ctli_set_param_fn set_param;
    ctli_audio_create_fn audio_create;
    ctli_audio_delete_fn audio_delete;
    ctli_audio_receive_fn audio_receive;
    ctli_audio_abort_fn audio_abort;
    ctli_audio_get_params_fn audio_get_params;
    ctli_audio_set_params_fn audio_set_params;
    ctli_dataport_select_fn dataport_select;
    ctli_dataport_send_fn dataport_send;
    ctli_dataport_recv_fn dataport_recv;
}

```

```
ctli_dataport_cancel_fn dataport_cancel;
ctli_dataport_enable_fn dataport_enable;
ctli_dataport_get_param_fn dataport_get_param;
ctli_dataport_set_param_fn dataport_set_param;
} ctli_context;

typedef int (*ctli_create_transport_fn)(const struct ctli_create_params* params, struct
ctli_context* ctx);

#endif // CINEMO_TRANSPORT_H_INCLUDED
```

HUMAX AUTOMOTIVE CONFIDENTIAL

6.2.1 Create_Transport_Custom

The create function will always be called by Cinemo if a custom transport needs to be established. It needs to have the signature of the *ctli_create_transport_fn* function pointer.

Syntax

```
int Create_Transport_Custom(
    const struct ctli_create_params * params,
    struct ctli_context* ctx
);
```

Parameters

- **params**

Pointer to *ctli_create_params* structure.

- **ctx**

Pointer to *ctli_context* structure.

Return value

If there is no error, the create function shall return *CTLI_NO_ERROR*.

If the URL cannot be handled by the transport implementation the create function shall return *CTLI_ERROR_NOTIMPLEMENTED*.

If any of the requested dataport cannot be supported the create function shall return *CTLI_ERROR_CONFIGURATION*.

In case of a fatal error the create function shall return *CTLI_ERROR_RESOURCES*.

In case of a non-fatal error the create function shall return any of the following error: *CTLI_ERROR_ARGUMENTS*, *CTLI_ERROR_IO*, *CTLI_ERROR_NOTCONNECTED*, *CTLI_ERROR_OPEN*.

All remaining errors shall not be returned by the create function.

Remarks

After returning from this function, the transport library must have finished the setup of the underlying transport and be prepared to receive any function call provided by the API.

The *ctli_create_params* structure is defined as follows:

```
struct ctli_create_params {
    const char * szurl;
    const char * szoption;
    const char * szdataports;
    ctli_hid_report_descriptor * phid;
    unsigned int hid_count;
};
```

The device URL passed in the create parameters will be exactly the same which has been passed to the Open() call of the ICinemolAP, ICinemoVFS or ICinemoPlaylist command, but stripped of the device protocol, e.g. "iap://". Cinemo will not try to interpret this URL.

During the Create() call the transport library needs to fill in all function pointers in the *ctli_context* structure. Cinemo will call these functions on demand and in no particular order.

All functions which are not supported by the transport layer shall return *CTLI_ERROR_NOTIMPLEMENTED*. If a transport does not support audio, all audio related functions shall also return *CTLI_ERROR_NOTIMPLEMENTED*.

General dataport setup

All dataports are assigned an internal ID implicitly, based on the zero-based index of their position in the *szdataports* string. This ID will be used in all dataport related function calls like [*ctli_dataport_select_fn\(\)*](#), [*ctli_dataport_send_fn\(\)*](#), [*ctli_dataport_recv_fn\(\)*](#), [*ctli_dataport_cancel_fn*](#) and [*ctli_dataport_enable_fn*](#).

Dataport setup for iAP devices

The *szoption* parameter will be set to the combination of the option strings provided in the [*ICinemolAP::Open\(\)*](#) call and in the [*ICinemoConfig*](#) object.

The *szdataports* parameter will be a comma separated list of dataports, generated from the device configuration metapool passed to the [*ICinemolAP::Open\(\)*](#) call. Each item in this list will contain a dataport type and an associated custom string. Currently there are three types of dataports:

- **neap**

Native EA protocol dataport. The custom string contains the protocol name, which needs to be set in the USB Interface Descriptor. Native EAP is only supported for USB host mode.

- **pneap**

Passive native EA protocol dataport. The difference for neap and pneap is that passive ports will not be handled by Cinemo. Instead any other process may claim these endpoints and handle them on their own. This can be triggered with the *CINEMO_METANAME_IAP_EAP_WATCH_NATIVE_SESSION* metapool entry for a native EA protocol in the iAP identification.

- **nhid**

Native HID dataport. The custom string will contain the HID interface number for USB components or "bluetooth" for BT components.

If there are any native HID components, the associated HID report descriptors will be passed in the *phid* and *hid_count* parameters in the same order as in the *szdataports* string.

6.2.2 protocol_version

The *protocol_version* should always be set to *CINEMO_TRANSPORT_LIBRARY_PROTOCOL_VERSION* from the corresponding interface header during the Create() function.

Cinemo will try to match the protocol version of the library with the internally used protocol version. If any incompatibility is detected, the transport will not be used and be deleted immediately.

6.2.3 `ctli_handle`

The `ctli_handle` is a void pointer available in the `ctli_context` structure. It will never be evaluated by Cinemo and may be used for internal data of the transport library.

All functions of the transport library receive the transport handle, which can be set in the Create() call, as their first argument.

6.2.4 `ctli_delete_transport_fn`

The `ctli_delete_transport_fn` function will be called if a transport needs to be shut down.

Syntax

```
int Delete_Transport_Custom(
    ctli_handle tphandle
);
```

Parameters

- **tphandle**

Transport handle of the transport to be deleted.

Return value

The return value of this function is ignored, Cinemo will assume the transport has been properly deleted.

Remarks

It must not be assumed that the Create() function will no longer be called after this call. Cinemo SDK may need to open a different device or reopen a previously closed device.

If [ctli_delete_transport_fn\(\)](#) is called, it is guaranteed that no other blocking function is currently being called (Receive / Send / DataportSelect / DataportSend / DataportRecv).

6.2.5 `ctli_receive_fn`

The `ctli_receive_fn` function will be called if Cinemo needs to receive data from the device.

Syntax

```
int Receive_Custom(
    ctli_handle tphandle,
    void* pdest,
    unsigned int npdest
);
```

Parameters

- **tphandle**

Transport handle.

- **pdest**

Pointer to receive buffer.

- **npdest**

Size of receive buffer in bytes.

Return value

If no error occurred, the function shall return the number of bytes available in the buffer. A negative return value, i.e. a `CTLI_ERROR`, indicates an error.

Remarks

The `ctli_receive_fn()` function needs to be implemented as a blocking function. It shall only return if data has been read from the device, an error occurred or `ctli_abort_fn()` has been called.

6.2.6 `ctli_send_fn`

The `ctli_send_fn` function will be called if Cinemo needs to send data to the device.

Syntax

```
int Send_Custom(  
    ctli_handle tphandle,  
    const void* psrc,  
    unsigned int npsrc  
) ;
```

Parameters

- **tphandle**

Transport handle.

- **psrc**

Pointer to send buffer.

- **npsrc**

Size of send buffer in bytes.

Return value

If no error occurred, the function shall return the number of bytes sent to the device. A negative return value, i.e. a `CTLI_ERROR`, indicates an error.

Remarks

The [ctli_send_fn\(\)](#) function needs to be implemented as a blocking function. It shall only return if data has been sent to the device, an error occurred or [ctli_abort_fn\(\)](#) has been called.

6.2.7 `ctli_abort_fn`

The `ctli_abort_fn` function will be called if Cinemo needs to unblock all currently blocking functions.

Syntax

```
int Abort_Custom(
    ctli_handle tphandle
);
```

Parameters

- **tphandle**

Transport handle.

Return value

The return value of this function is ignored, Cinemo will assume that all blocking functions have been unblocked.

Remarks

The [ctli_abort_fn\(\)](#) function needs to unblock the [ctli_receive_fn\(\)](#), [ctli_send_fn\(\)](#), [ctli_dataport_select_fn\(\)](#), [ctli_dataport_send_fn\(\)](#) and [ctli_dataport_receive_fn\(\)](#) functions. The state of the transport library shall be set to disabled and all subsequent function calls shall return `CTLI_ERROR_CANCEL`.

After [ctli_abort_fn\(\)](#) has been called this transport library instance will be deleted and not used for any further device communication.

Audio related functions which may be blocking are unblocked with [ctli_audio_abort_fn\(\)](#).

6.2.8 `ctli_get_param_fn`

The `ctli_get_param_fn` function will be called if Cinemo needs to retrieve any transport related parameter.

Syntax

```
int Get_Param_Custom(  
    ctli_handle tphandle,  
    CTLI_PARAM param_type,  
    ctli_params* params  
) ;
```

Parameters

- **tphandle**

Transport handle.

- **param_type**

`CTLI_PARAM` enum.

- **params**

Pointer to `ctli_params` union.

Return value

If there is no error the function shall return `CTLI_NO_ERROR`. A negative return value, i.e. a `CTLI_ERROR`, indicates an error.

Remarks

The [`ctli_get_param_fn\(\)`](#) function may be called anytime by Cinemo to retrieve information about the underlying transport layer.

If the requested parameter is unknown or unavailable to the transport layer `CTLI_ERROR_NOTIMPLEMENTED` shall be returned.

All possible parameters are defined in the `CTLI_PARAM` enum.

```
typedef enum CTLI_PARAM {  
    CTLI_PARAM_PROTOCOL,  
    CTLI_PARAM_STATE,  
    CTLI_PARAM_USB_VID,  
    CTLI_PARAM_USB_PID,  
    CTLI_PARAM_USB_MANUFACTURER,  
    CTLI_PARAM_USB_PRODUCT,  
    CTLI_PARAM_USB_SERIALNO,  
    CTLI_PARAM_NCM_IF_NUM,  
    CTLI_PARAM_BLUETOOTH_MAC_ADDRESS,  
    CTLI_PARAM_BLUETOOTH_ADAPTER_NAME,  
    CTLI_PARAM_BLUETOOTH_DEVICE_NAME,  
    CTLI_PARAM_BLUETOOTH_AVRCP_VERSION,
```

```
    CTLI_PARAM_BLUETOOTH_AVRCP_FEATURESFLAGS,  
    CTLI_PARAM_BLUETOOTH_AVRCP_BROWSING_PSM,  
    CTLI_PARAM_BLUETOOTH_L2CAP_IMTU,  
    CTLI_PARAM_BLUETOOTH_L2CAP_OMTU  
} CTLI_PARAM;
```

- **CTLI_PARAM_PROTOCOL**

Communication protocol – enum *CTLI_PARAM_PROTOCOL_VALUES*

- **CTLI_PARAM_STATE**

USB mode – enum *CTLI_PARAM_STATE_VALUES*

- **CTLI_PARAM_USB_VID**

USB vendor ID

- **CTLI_PARAM_USB_PID**

USB product ID

- **CTLI_PARAM_USB_MANUFACTURER**

USB manufacturer

- **CTLI_PARAM_USB_PRODUCT**

USB product

- **CTLI_PARAM_USB_SERIALNO**

USB serial number

- **CTLI_PARAM_NCM_IF_NUM**

NCM interface number

- **CTLI_PARAM_BLUETOOTH_MAC_ADDRESS**

MAC address of the Bluetooth adapter

- **CTLI_PARAM_BLUETOOTH_ADAPTER_NAME**

Name of the Bluetooth adapter

- **CTLI_PARAM_BLUETOOTH_DEVICE_NAME**

Name of the remote Bluetooth device

- **CTLI_PARAM_BLUETOOTH_AVRCP_VERSION**

AVRCP version of the device (e.g. 0x0104 => AVRCP 1.4)

- **CTLI_PARAM_BLUETOOTH_AVRCP_FEATURESFLAGS**

Features flag declared in the SDP for AVRCP

- **CTLI_PARAM_BLUETOOTH_AVRCP_BROWSING_PSM**

PSM of the optional browsing connection declared in SDP for AVRCP

- **CTLI_PARAM_BLUETOOTH_L2CAP_IMTU**

MTU for receiving packets

- **CTLI_PARAM_BLUETOOTH_L2CAP_OMTU**

MTU for sending packets

6.2.9 `ctli_set_param_fn`

The `ctli_set_param_fn` function will be called if Cinemo needs to set certain transport layer parameters.

Syntax

```
int Set_Param_Custom(  
    ctli_handle tphandle,  
    CTLI_PARAM param_type,  
    const ctli_params* params  
) ;
```

Parameters

- **tphandle**

Transport handle.

- **param_type**

`CTLI_PARAM` enum.

- **params**

Pointer to `ctli_params` union.

Return value

If there is no error the function needs to return `CTLI_NO_ERROR`. A negative return value, i.e. a `CTLI_ERROR`, indicates an error.

Remarks

The [`ctli_get_param_fn\(\)`](#) function may be called at different points in time to set parameters required by the underlying transport layer.

Currently this function is not used.

6.2.10 `ctli_audio_create_fn`

The `ctli_audio_create_fn` function will be called if Cinemo needs to start streaming audio data via this transport.

Syntax

```
int Audio_Create_Custom(
    ctli_handle tphandle
);
```

Parameters

- **tphandle**

Transport handle.

Return value

If there is no error the function needs to return `CTLI_NO_ERROR`. A negative return value indicates an error.

Remarks

The [`ctli_audio_create_fn\(\)`](#) function needs to set up the transport layer for transferring audio data. It will be called before any other audio related functions are called.

6.2.11 `ctli_audio_delete_fn`

The `ctli_audio_delete_fn` function will be called if Cinemo needs to end audio data streaming via the transport.

Syntax

```
int Audio_Delete_Custom(
    ctli_handle tphandle
);
```

Parameters

- **tphandle**

Transport handle.

Return value

The return value of this function is ignored, Cinemo will assume that all audio transfers have been deleted.

Remarks

It must not be assumed that closing the audio part of the transport will also shut down the data transfer. Only audio related transport functionality may be suspended after this call.

If [ctli_audio_delete_fn\(\)](#) is called, it is guaranteed that no other blocking audio function is currently being called (AudioReceive).

6.2.12 `ctli_audio_receive_fn`

The `ctli_audio_receive_fn` function will be called if Cinemo needs to receive audio data from the device.

Syntax

```
int Audio_Receive_Custom(  
    ctli_handle tphandle,  
    void* pdest,  
    unsigned int npdest  
) ;
```

Parameters

- **tphandle**

Transport handle.

- **pdest**

Pointer to audio receive buffer.

- **npdest**

Size of audio receive buffer in bytes.

Return value

If there is no error, the function needs to return the number of bytes available in the audio buffer. A negative return value indicates an error.

Remarks

The [ctli_audio_receive_fn\(\)](#) function needs to be implemented as a blocking function. It shall only return if data has been read from the device, an error occurred or [ctli_audio_abort_fn\(\)](#) has been called.

If new data about audio parameters is available from the underlying device, this function needs to return with `CTLI_ERROR_AUDIOPARAMS` (defined in the Cinemo transport library header file). In this case Cinemo will request the new audio parameters from the transport library and apply them to the audio pipeline.

If the audio stream has been suspended temporarily, e.g. for a BT connection, this function shall return `CTLI_ERROR_IDLE` to notify Cinemo about this state. After returning the error, the receive function will be called again immediately by Cinemo and needs to block until the audio stream is enabled again or is terminated finally.

6.2.13 `ctli_audio_abort_fn`

The `ctli_audio_abort_fn` function will be called if Cinemo needs to unblock any audio-related function calls.

Syntax

```
int Audio_Abort_Custom(  
    ctli_handle tphandle  
) ;
```

Parameters

- **tphandle**

Transport handle.

Return value

The return value of this function is ignored.

Remarks

If this function is called the [`ctli_audio_receive_fn\(\)`](#) function shall be unblocked.

6.2.14 `ctli_audio_get_params_fn`

The `ctli_audio_get_params_fn` function will be called if Cinemo needs to receive the audio parameters from the transport layer.

Syntax

```
int Audio_Get_Parms_Custom(
    ctli_handle tphandle,
    ctli_audio_params* params
);
```

Parameters

- **tphandle**

Transport handle.

- **params**

Pointer to audio parameters.

Return value

If there is no error the function must return `CTLI_NO_ERROR`. A negative return value indicates an error.

Remarks

The `ctli_audio_get_params_fn()` function needs to be implemented only for transports which provide information about audio parameters (e.g. iAP for devices operating in USB host mode).

The `ctli_audio_params` structure is defined as follows:

```
struct ctli_audio_params {
    CTLI_AUDIO_CONTAINER container;
    CTLI_AUDIO_CODEC codec;
    CTLI_AUDIO_SAMPLETYPE samplertype;
    int samplerate;
    int channels;
    int bits;
};
```

The `CTLI_AUDIO_SAMPLETYPE` enum is defined as

```
enum CTLI_AUDIO_SAMPLETYPE {
    CTLI_AUDIO_SAMPLETYPE_UNSPECIFIED,
    CTLI_AUDIO_SAMPLETYPE_S16LE,
    CTLI_AUDIO_SAMPLETYPE_S24LE
};
```

6.2.15 `ctli_audio_set_params_fn`

The `ctli_audio_set_params_fn` function will be called if Cinemo needs to set audio parameters to the transport layer.

Syntax

```
int Audio_Set_Parms_Custom(  
    ctli_handle tphandle,  
    ctli_audio_params* params  
) ;
```

Parameters

- **tphandle**

Transport handle.

- **params**

Pointer to audio parameters.

Return value

If there is no error the function must return `CTLI_NO_ERROR`. A negative return value indicates an error.

Remarks

The `ctli_audio_set_params_fn()` function needs to be implemented only for transports which require information about audio parameters (e.g. iAP for devices operating in USB device mode).

6.2.16 `ctli_dataport_select_fn`

The `ctli_dataport_select_fn` function will be called by Cinemo to be notified about dataport state changes.

Syntax

```
int Dataport_Select_Custom(  
    ctli_handle tphandle,  
    ctli_dataport_state * state,  
    unsigned int num_entries  
) ;
```

Parameters

- **tphandle**

Transport handle.

- **state**

Pointer `ctli_dataport_state` structure.

- **num_entries**

Number of state entries.

Return value

If there is no error the function must return `CTLI_NO_ERROR`. A negative return value indicates an error.

Remarks

This method shall be implemented like the POSIX select() call. It receives a `ctli_dataport_state` structure and shall block until any dataport from this structure changed its state. Currently this is only used for NEAP dataports.

This method shall be unblocked if [ctli_abort_fn\(\)](#) is called.

6.2.17 `ctli_dataport_send_fn`

The `ctli_dataport_send_fn` function will be called when Cinemo needs to send data to any connected dataport.

Syntax

```
int Dataport_Send_Custom(
    ctli_handle tphandle,
    unsigned short id,
    const void* psrc,
    unsigned int npsrc
);
```

Parameters

- **tphandle**

Transport handle.

- **id**

Dataport ID.

- **psrc**

Pointer to data.

- **npsrc**

Size of data.

Return value

If there is no error the function must return the number of bytes sent. A negative return value indicates an error from the defined set of `CTLI_ERROR` constants. If the dataport with the provided ID is in “canceled state”, then the function must return `CTLI_ERROR_CANCEL`.

Remarks

This method will be called whenever Cinemo needs to send data to the dataport associated with the provided ID. The Dataport ID corresponds to the index of the dataport in the `szdataports` string passed to the [ctli_create_transport_fn\(\)](#) function.

This method shall be unblocked if [ctli_abort_fn\(\)](#) is called.

This method shall be unblocked if [ctli_dataport_cancel_fn\(\)](#) was called for the same data port ID as in the current invocation. If a partial amount of data has been sent already, the function shall first return “successful” with the current number of sent bytes. In the next invocation it shall then return `CTLI_ERROR_CANCEL`.

6.2.18 `ctli_dataport_recv_fn`

The `ctli_dataport_recv_fn` function will be called when Cinemo needs to receive data from any connected dataport.

Syntax

```
int Dataport_Recv_Custom(
    ctli_handle tphandle,
    unsigned short id,
    void* pdest,
    unsigned int npdest
);
```

Parameters

- **tphandle**

Transport handle.

- **id**

Dataport ID.

- **pdest**

Pointer to receive buffer.

- **npdest**

Size of receive buffer.

Return value

If there is no error the function must return the number of bytes received. A negative return value indicates an error from the defined set of `CTLI_ERROR` constants. If the dataport with the provided ID is in “canceled state”, then the function must return `CTLI_ERROR_CANCEL`.

Remarks

This method will be called whenever Cinemo needs to receive data from the dataport associated with the provided ID. The Dataport ID corresponds to the index of the dataport in the `szdataports` string passed to the [ctli_create_transport_fn\(\)](#) function.

If there is no data to be received the function blocks and waits until some data is available.

This method shall be unblocked if [ctli_abort_fn\(\)](#) is called.

This method shall be unblocked if [ctli_dataport_cancel_fn\(\)](#) was called for the same data port ID as in the current invocation. If a partial amount of data has been received already, the function shall first return “successful” with the current number of received bytes. In the next invocation it shall then return `CTLI_ERROR_CANCEL`.

6.2.19 `ctli_dataport_cancel_fn`

The `ctli_dataport_cancel_fn` function will be called when a blocking dataport operation needs to be canceled.

Syntax

```
int Dataport_Cancel_Custom(  
    ctli_handle tphandle,  
    unsigned short id  
) ;
```

Parameters

- **tphandle**

Transport handle.

- **id**

Dataport ID.

Return value

If there is no error the function must return `CTLI_NO_ERROR`. A negative return value indicates an error.

Remarks

This method may be called to asynchronously cancel any thread blocked on `ctli_dataport_send_fn()` or `ctli_dataport_recv_fn()` calls. In this case, `ctli_dataport_send_fn()` or `ctli_dataport_recv_fn()` shall return `CTLI_ERROR_CANCEL`.

The Dataport ID corresponds to the index of the dataport in the `szdataports` string passed to the `ctli_create_transport_fn()` function.

Note the cancel operation is sticky – after calling `ctli_dataport_cancel_fn()`, the associated dataport shall remain in a canceled state until `ctli_dataport_enable_fn()` is called. This is to avoid potential race conditions.

6.2.20 `ctli_dataport_enable_fn`

The `ctli_dataport_enable_fn` function will be called if a dataport shall be re-enabled after being canceled.

Syntax

```
int Dataport_Select_Custom(  
    ctli_handle tphandle,  
    unsigned short id  
) ;
```

Parameters

- **tphandle**

Transport handle.

- **id**

Dataport ID.

Return value

If there is no error the function must return `CTLI_NO_ERROR`. A negative return value indicates an error.

Remarks

This method may be called to enable a particular dataport which has been canceled previously.

The Dataport ID corresponds to the index of the dataport in the `szdataports` string passed to the [ctli_create_transport_fn\(\)](#) function.

Note the cancel operation is sticky – after calling [ctli_dataport_cancel_fn\(\)](#), the associated dataport shall remain in a canceled state until [ctli_dataport_enable_fn\(\)](#) is called. This is to avoid potential race conditions.

6.2.21 `ctli_dataport_get_param_fn`

The `ctli_dataport_get_param_fn` function will be called if Cinemo needs to retrieve any dataport related transport parameter.

Syntax

```
int Dataport_Get_Param_Custom(
    ctli_handle tphandle,
    unsigned short id,
    CTLI_DATAPORT_PARAM param_type,
    ctli_dataport_params* params
);
```

Parameters

- **tphandle**

Transport handle.

- **id**

Dataport ID.

- **param_type**

`CTLI_DATAPORT_PARAM` enum.

- **params**

Pointer to `ctli_dataport_params` union.

Return value

If there is no error the function must return `CTLI_NO_ERROR`. A negative return value indicates an error.

Remarks

The `ctli_dataport_get_param_fn()` function may be called anytime by Cinemo to retrieve information about a specific dataport.

If the requested parameter is unknown or unavailable to the transport layer `CTLI_ERROR_NOTIMPLEMENTED` shall be returned.

Currently this function is not used.

6.2.22 `ctli_dataport_set_param_fn`

The `ctli_dataport_set_param_fn` function will be called if Cinemo needs to set any dataport related transport parameter.

Syntax

```
int Dataport_Set_Param_Custom(
    ctli_handle tphandle,
    unsigned short id,
    CTLI_DATAPORT_PARAM param_type,
    const ctli_dataport_params* params
);
```

Parameters

- **tphandle**

Transport handle.

- **id**

Dataport ID.

- **param_type**

`CTLI_DATAPORT_PARAM` enum.

- **params**

Pointer to `ctli_dataport_params` union.

Return value

If there is no error the function must return `CTLI_NO_ERROR`. A negative return value indicates an error.

Remarks

The `ctli_dataport_set_param_fn()` function may be called anytime by Cinemo to set specific dataport parameters.

If the requested parameter is unknown or unavailable to the transport layer `CTLI_ERROR_NOTIMPLEMENTED` shall be returned.

All possible parameters are defined in the `CTLI_DATAPORT_PARAM` enum.

```
typedef enum CTLI_DATAPORT_PARAM {
    CTLI_DATAPORT_PARAM_STATE = 0,
} CTLI_DATAPORT_PARAM;
```

- **CTLI_DATAPORT_PARAM_STATE**

Current state of the dataport – enum `CTLI_DATAPORT_STATE`

6.3 Error Codes

The Cinemo Transport Library uses error codes defined in the *cinemo_transport.h* header file as return values for all functions. If any other value is returned by any function it will be regarded as a fatal error and the transport will be terminated.

The cases in which certain functions may return any of the error codes is described in the following sections.

6.3.1 CTLI_NO_ERROR

This is the default return value for all functions which were executed successfully.

6.3.2 CTLI_ERROR_ARGUMENTS

This error shall be returned by all functions if any arguments is invalid or missing.

6.3.3 CTLI_ERROR_AUDIOPARAMS

This error shall only be returned by the [ctli_audio_receive_fn\(\)](#) function. It indicates that new audio parameters are available from the transport layer. Audio data shall not be passed if this error is returned. After the audio receive function returns CTLI_ERROR_AUDIOPARAMS Cinemo will read the new parameters via [ctli_audio_get_params_fn\(\)](#) and continue reading audio data afterwards.

6.3.4 CTLI_ERROR_CANCEL

This error shall be returned by any transport library function after [ctli_abort_fn\(\)](#) or [ctli_audio_abort_fn\(\)](#) have been called. All functions which have been unblocked by a cancel call shall also return with CTLI_ERROR_CANCEL.

6.3.5 CTLI_ERROR_DECODE

This error shall only be returned for iAP-based transport implementations in the case when HID reports from the device cannot be decoded properly.

6.3.6 CTLI_ERROR_IO

This error shall be returned if a read/write operation failed due to a system related error.

6.3.7 CTLI_ERROR_NOTCONNECTED

This error shall be returned if the device is not or no longer connected.

6.3.8 CTLI_ERROR_NOTENOUGHSPACE

This error shall be returned if the buffer for the [ctli_get_param_fn\(\)](#) function has an insufficient capacity.

6.3.9 CTLI_ERROR_NOTIMPLEMENTED

This error shall only be returned by the following functions:

- **create function**

If the URL is not supported by this transport

- **ctli_get_param_fn**

If the CTLI_PARAM is not supported/available for this transport

- **ctli_audio_create_fn**

If no audio implementation is available for this transport

6.3.10 CTLI_ERROR_OPEN

This error shall be returned if the device or any other required file cannot be opened.

6.3.11 CTLI_ERROR_OVERFLOW

This error shall be returned if the buffer provided to [ctli_receive_fn\(\)](#), [ctli_audio_receive_fn\(\)](#) or [ctli_dataport_recv_fn\(\)](#) is too small.

6.3.12 CTLI_ERROR_RESOURCES

This error shall be returned if there is any problem with system resources, e.g. the system is out of memory.

6.3.13 CTLI_ERROR_AUDIOSETUP

This error shall be returned if the parameters for the audio stream have not been configured but the audio receive function has been called.

6.3.14 CTLI_ERROR_CONFIGURATION

This error shall be returned by the create function if the dataports requested in the *ctli_create_params* structure cannot be supported by this transport.

6.3.15 CTLI_ERROR_IDLE

This error shall be returned once by the audio receive function if the audio stream has been suspended temporarily.

6.3.16 CTLI_ERROR_UNAVAILABLE

The error shall be returned by the [ctli_audio_get_params_fn\(\)](#) if the audio format is unknown.

6.3.17 CTLI_ERROR_DRIVER_FAILURE

This special error code may only be returned by any function after consultation with Cinemo and should not be used in general.

6.4 Cinemo SDK options

Cinemo SDK currently provides two options related to the Cinemo transport library.

6.4.1 Transport Library Implementations

Currently transport implementations available as dynamic or static libraries can be used in the Cinemo transport layer.

Dynamic libraries

When creating a new transport for accessing an external device, Cinemo SDK will read the respective transport libraries option. It may contain a comma separated list of transport implementations, which will be used in the order they are given. If one implementation fails, the next one will be used.

Currently there are the following options:

- *CINEMO_OPTION_IAP_TRANSPORT_LIBRARIES*

Each entry in this option needs to contain two pieces of information, separated by a colon. First the name of the shared library needs to be specified, then the name of the Create() function.

For a Create() function called “CreateCustomTransport”, which is available in a shared library with the name “CustomTransport” the option should look like this:

```
CustomTransport:CreateCustomTransport
```

a second implementation (from the same library) may be added like:

```
CustomTransport:CreateCustomTransport,CustomTransport:CreateDeprecatedTransport
```

Shared library names will be converted to OS specific names. For the above example Cinemo SDK would try to access “libCustomTransport.so” on a Linux operating system.

All libraries need to be available in the *CINEMO_PLUGIN_INIT* path.

Static libraries

It is also possible to use static libraries with the Cinemo transport layer. The binary library needs to be provided to your Cinemo representative, who will adjust the transport library string individually to work with static linking.

6.4.2 Transport Library Option String

The option string passed when a new transport is created is generated from two sources. There are local transport options which can be specified each time a new device is created and there are global options which will be used for all transports. Local options will always override global options if they are defined in both strings.

All options need to be passed as URL parameters. Cinemo will internally merge both strings and pass the result on to the individual transport library implementations.

Local transport option string

Local transport options need to be passed when creating a new device. For iAP devices this can be achieved in the CinemoIAPConfig structure of ICinemolAP::Open(). This option string may be used to specify special settings for each device.

Global transport option string

Global transport options may be defined in the *CINEMO_OPTION_IAP_TRANSPORT_OPTION_STRING* for iAP.

Supported Transport Library option string parameters for iAP

The desired USB mode may be specified for all USB-based transports with the “usb_mode” parameter. It supports the values “host” or “device” and forces the connected iAP device in the respective state (if supported). Any other value, or not setting this parameter, will always use the transport implementations in the order they are specified in the *CINEMO_OPTION_IAP_TRANSPORT_LIBRARIES* option.

The CreateNmeIAPOTGTransport implementation currently supports the following parameters:

- **otg_device_regex**
Regular expression for acceptable USB device path.
- **otg_driver**
Path of the USB mode switching device.
- **ffs_path**
Mount path of the FunctionFS.
- **wait_disconnect**
If set to 1, the transport will wait after sending the role switch command to the device until the device has been disconnected from the USB bus.

The CreateNmeIAPFFSTransport implementation currently supports the following parameters:

- ncm_interface

A valid global option string might look like this:

```
?usb_mode=host&otg_device_regex=^/dev/bus/usb/001/\d+$/&otg_driver=/dev/otg_drv
```

When opening a new device with the local options string

```
?usb_mode=device
```

the local option will take precedence over the global option, resulting in the following option string:

```
?usb_mode=device&otg_device_regex=^/dev/bus/usb/001/\d+$/&otg_driver=/dev/otg_drv
```

The “enable_carplay” parameter will reflect the state of the *CINEMO_IAP_OPEN_FOR_CARPLAY* flag passed to the ICinemolAP::Open() call. If the flag is set, “enable_carplay” will be set to 1, otherwise it will be set to 0, indicating USB host mode should be enabled without the CarPlay capability.

HUMAX Confidential

7. Cinemo Apple Authentication Library

The Cinemo Apple Authentication Library interface is defined in a C header file which is provided by your Cinemo representative.

7.1 General Remarks

Cinemo provides an interface for accessing an Apple authentication chip with a custom interface. In this document the interface will be described and important correlations of the individual parts are explained.

7.1.1 Requirements

A custom implementation of the Cinemo Apple authentication library has to fulfill the following requirements:

- It must be binary compatible with regular Cinemo SDK releases for the target platform.
- It must not throw any exceptions.

If these requirements are met, Cinemo will be able to use the Apple authentication library implementation for accessing the authentication chip.

If the Apple authentication library is provided to Cinemo, either as binary or as source code, it is possible to include it in a release of the Cinemo SDK.

7.2 Apple Authentication Interface

Apple Authentication Interface Methods

METHOD	DESCRIPTION
<u>Create_AppleAuth_Custom</u>	Create function
<u>protocol_version</u>	Protocol version of this implementation
<u>cali_handle</u>	Custom Apple authentication handle
<u>cali_delete_authentication_fn</u>	Delete the Apple authentication instance
<u>cali_get_serial_number_fn</u>	Get the serial number
<u>cali_get_certificate_fn</u>	Get the certificate
<u>cali_get_challenge_response_fn</u>	Get the challenge response

The interface for the Cinemo Apple authentication library is provided in the form of a header file.

```
*****
Project:      Cinemo Media Engine

Developed by:  Cinemo GmbH
               Karlsruhe, Germany

Copyright (c) 2009-2017 Cinemo GmbH and its licensors.
All rights reserved.

This software is supplied only under the terms of a license agreement,
nondisclosure agreement or other written agreement with Cinemo GmbH.
Use, redistribution or other disclosure of any parts of this software
is prohibited except in accordance with the terms of such written
agreement with Cinemo GmbH. This software is confidential and proprietary
information of Cinemo GmbH and its licensors.

In case of licensing questions please contact: sales@cinemo.com
***** */

#ifndef CINEMO_APPLE_AUTH_INTERFACE_H_INCLUDED
#define CINEMO_APPLE_AUTH_INTERFACE_H_INCLUDED

#define CINEMO_AUTHENTICATION_LIBRARY_PROTOCOL_VERSION 1

#define CALI_NO_ERROR          0
#define CALI_ERROR_ARGUMENTS   -1
#define CALI_ERROR_IO           -2
#define CALI_ERROR_NOTIMPLEMENTED -3
#define CALI_ERROR_OPEN          -4
#define CALI_ERROR_OVERFLOW     -5
#define CALI_ERROR_RESOURCES    -6

typedef struct cali_create_params {
```

```
const char* szurl;
const char* szoption;
} cali_create_params;

typedef void* cali_handle;

typedef int (*cali_delete_authentication_fn)(cali_handle auth_handle);
typedef int (*cali_get_serial_number_fn)(cali_handle auth_handle, void* pdest, unsigned int npdest);
typedef int (*cali_get_certificate_fn)(cali_handle auth_handle, void* pdest, unsigned int npdest);
typedef int (*cali_get_challenge_response_fn)(cali_handle auth_handle, void* pdest, unsigned int npdest, const void* pchallenge, unsigned int nbytes);

typedef struct cali_context {
    unsigned short protocol_version;
    cali_handle auth_handle;
    cali_delete_authentication_fn delete_authentication;
    cali_get_serial_number_fn get_serial_number;
    cali_get_certificate_fn get_certificate;
    cali_get_challenge_response_fn get_challenge_response;
} cali_context;

typedef int (*cali_create_apple_auth_fn)(const struct cali_create_params* params, struct cali_context* ctx);

#endif // CINEMO_APPLE_AUTH_INTERFACE_H_INCLUDED
```

7.2.1 Create_AppleAuth_Custom

The create function will be called by Cinemo if an instance of the Apple authentication library needs to be created. The signature needs to match the *cali_create_apple_auth_fn* function pointer.

Syntax

```
int Create_AppleAuth_Custom(
    const struct cali_create_params * params,
    struct cali_context * ctx
);
```

Parameters

- **params**

Pointer to *cali_create_params* structure.

- **ctx**

Pointer to *cali_context* structure.

Return value

If there is no error, the create function shall return *CALI_NO_ERROR*.

If the URL cannot be handled by the implementation, the create function shall return *CALI_ERROR_NOTIMPLEMENTED*.

In case of a fatal error the create function shall return *CALI_ERROR_RESOURCES*.

In case of a non-fatal error the create function shall return any of the following error: *CALI_ERROR_ARGUMENTS*, *CALI_ERROR_IO*, *CALI_ERROR_OPEN*.

All remaining errors shall not be returned by the create function.

Remarks

After returning from this function, the Apple authentication library must have finished the initialization of the authentication chip and be prepared to receive any function call provided by the API.

The *cali_create_params* structure is defined as follows:

```
struct cali_create_params {
    const char * szurl;
    const char * szoption;
} cali_create_params;
```

The device URL passed in the create parameters will be exactly the same which has been passed to the [ICinemoAppleAuth::Open\(\)](#) command. Cinemo will not try to interpret this URL.

The *szoption* parameter will be set to the combination of the option strings provided in the [ICinemoAppleAuth::Open\(\)](#) call and in the [ICinemoConfig](#) object.

During the Create() call the Apple authentication library needs to fill in all function pointers in the *cali_context* structure. Cinemo will call these functions on demand and in no particular order.

HUMAX Confidential

7.2.2 protocol_version

The *protocol_version* should always be set to *CINEMO_AUTHENTICATION_LIBRARY_PROTOCOL_VERSION* from the corresponding interface header during the Create() function.

Cinemo will try to match the protocol version of the library with the internally used protocol version. If any incompatibility is detected, the authentication implementation will not be used and be deleted immediately.

7.2.3 cali_handle

The *cali_handle* is a void pointer available in the *cali_context* structure. It will never be evaluated by Cinemo and may be used for internal data of the Apple authentication library.

All functions of the Apple authentication library receive this handle, which can be set in the Create() call, as their first argument.

7.2.4 cali_delete_authentication_fn

The *cali_delete_authentication_fn* function will be called if the authentication instance shall be deleted.

Syntax

```
int Delete_AppleAuth_Custom(
    cali_handle auth_handle
);
```

Parameters

- **auth_handle**

Authentication handle of the instance to be deleted.

Return value

The return value of this function is ignored, Cinemo will assume the authentication library has been deleted properly.

Remarks

It must not be assumed that the Create() function will no longer be called after this call. Cinemo SDK may need to (re)open the Apple authentication chip depending on the circumstances.

If *cali_delete_authentication_fn* is called, it is guaranteed that no other blocking function is currently being called (GetCertificate/GetChallengeResponse).

7.2.5 cali_get_serial_number_fn

The *cali_get_serial_number_fn* function will be called if the authentication chip serial number is required.

Syntax

```
int Get_Serial_Number_Custom(  
    cali_handle auth_handle,  
    void * pdest,  
    unsigned int npdest  
) ;
```

Parameters

- **auth_handle**

Authentication handle of the instance.

- **pdest**

Pointer to receive buffer.

- **npdest**

Size of receive buffer in bytes.

Return value

If no error occurred, the function shall return the number of bytes available in the buffer. A negative return value, i.e. *CALI_ERROR*, indicates an error.

Remarks

If the Apple Authentication Coprocessor serial number is not available *CALI_ERROR_NOTIMPLEMENTED* shall be returned.

7.2.6 cali_get_certificate_fn

The *cali_get_certificate_fn* function will be called if the authentication chip certificate data is required.

Syntax

```
int Get_Certificate_Custom(  
    cali_handle auth_handle,  
    void * pdest,  
    unsigned int npdest  
) ;
```

Parameters

- **auth_handle**

Authentication handle of the instance.

- **pdest**

Pointer to receive buffer.

- **npdest**

Size of receive buffer in bytes.

Return value

If no error occurred, the function shall return the number of bytes available in the buffer. A negative return value, i.e. *CALI_ERROR*, indicates an error.

Remarks

This method shall return the complete certificate, which may consist of up to 10 parts. Returning a single certificate part is not supported.

7.2.7 cali_get_challenge_response_fn

The *cali_get_challenge_response_fn* function will be called if the challenge response is required.

Syntax

```
int Get_Challenge_Response_Custom(
    cali_handle auth_handle,
    void * pdest,
    unsigned int npdest,
    const void * pchallenge,
    unsigned int nbytes
);
```

Parameters

- **auth_handle**

Authentication handle of the instance.

- **pdest**

Pointer to receive buffer.

- **npdest**

Size of receive buffer in bytes.

- **pchallenge**

Pointer to authentication challenge data.

- **nbytes**

Size of authentication challenge data in bytes.

Return value

If no error occurred, the function shall return the number of bytes available in the receive buffer. A negative return value, i.e. *CALI_ERROR*, indicates an error.

Remarks

The provided challenge response needs to be transmitted to the Apple authentication chip, which will then return the calculated challenge response.

7.3 Error Codes

The Cinemo Apple Authentication Library uses error codes defined in the *cinemo_apple_auth_interface.h* header file as return values for all functions. If any other value is returned by any function it will be regarded as a fatal error and the authentication library will be terminated.

The cases in which certain functions may return any of the error codes is described in the following sections.

7.3.1 CALI_NO_ERROR

This is the default return value for all functions which were executed successfully.

7.3.2 CALI_ERROR_ARGUMENTS

This error shall be returned by all functions if any arguments is invalid or missing.

7.3.3 CALI_ERROR_IO

This error shall be returned if a read/write operation failed due to a system related error.

7.3.4 CALI_ERROR_NOTIMPLEMENTED

This error shall only be returned by the following functions:

- **create function**
If the URL is not supported by this authentication library.
- **cali_get_serial_number_fn**
If no serial number is available.

7.3.5 CALI_ERROR_OPEN

This error shall be returned if the device or any other required file cannot be opened.

7.3.6 CALI_ERROR_OVERFLOW

This error shall be returned if the receive buffer for authentication data is too small.

7.3.7 CALI_ERROR_RESOURCES

This error shall be returned if there is a system resource problem, e.g. the system is out of memory.

7.4 Cinemo SDK options

Cinemo SDK currently provides three options related to the Cinemo Apple Authentication library.

7.4.1 Apple Authentication Library URL

The URL for communication with the Apple Authentication Coprocessor. On QNX systems the slave address of the Authentication Coprocessor (16 or 17, as defined by Apple) may be required to be shifted left by one bit, for example: "i2c://dev/i2c1:34" for slave address 17.

For example, on Linux systems:

- `ICinemoConfig::SetOption(CINEMO_OPTION_APPLEAUTH_URL, "i2c://dev/i2c1:16")`

The default value for this option is usually set by Cinemo on a project basis.

7.4.2 Apple Authentication Library Implementations

Currently Apple Authentication library implementations available as dynamic or static libraries can be used in Cinemo.

Dynamic libraries

When creating a new Apple Authentication instance, Cinemo SDK will read the `CINEMO_OPTION_APPLEAUTH_LIBRARIES` option. It may contain a comma separated list of authentication implementations, which will be used in the order they are given. If one implementation fails, the next one will be used.

Each entry in this option needs to contain two pieces of information, separated by a colon. First the name of the shared library needs to be specified, then the name of the `Create()` function.

For a `Create()` function called "CreateCustomAppleAuth", which is available in a shared library with the name "CustomAppleAuth" the option should look like this:

```
CustomAppleAuth:CreateCustomAppleAuth
```

a second implementation (from the same library) may be added like:

```
CustomAppleAuth:CreateCustomAppleAuth,CustomAppleAuth:CreateDeprecatedAppleAuth
```

Shared library names will be converted to OS specific names. For the above example Cinemo SDK would try to access "libCustomAppleAuth.so" on a Linux operating system.

All libraries need to be available in the `CINEMO_PLUGIN_INIT` path.

Static libraries

It is also possible to use static libraries with the Cinemo Apple Authentication layer. The binary library needs to be provided to your Cinemo representative, who will adjust the authentication library string individually to work with static linking.

7.4.3 Apple Authentication Library Option String

The option string passed when a new authentication instance is created is generated from two sources. There are local options which can be specified when using the [ICinemoAppleAuth](#) interface and there are global options which will be used for instances created from inside Cinemo. Local options will always override global options if they are defined in both strings.

All options need to be passed as URL parameters. Cinemo will internally merge both strings and pass the result on to the individual authentication library implementations.

Local authentication option string

Local transport options need to be passed when creating a new [ICinemoAppleAuth](#) instance. This option string may be used to specify special settings for each device. However, if an internal instance with the same URL already exists, no new instance will be created and the local option string will not be used.

Global authentication option string

Global transport options may be defined in the `CINEMO_OPTION_APPLEAUTH_OPTION_STRING`.

Supported Apple Authentication Library option string parameters

The following options are supported:

- **i2c_delay_microsecs**

Delay in microseconds between two subsequent write and read operations when accessing the Apple Authentication Coprocessor.

- **i2c_write_extra_microsecs**

Additional wait time in microseconds after each write call to the Apple Authentication Coprocessor.

- **i2c_retry_count**

Maximum number of retries when read or write operations fail.

- **i2c_retry_delay_ms**

The delay between two subsequent retries in ms.

- **sclock (QNX only)**

The Authentication Coprocessor version 2.0B requires that the i2c bus speed cannot exceed 50 kHz. In order to support QNX systems where the bus speed might exceed 50 kHz, this option sets the bus speed.

- **txn (QNX only)**

If set to a non-zero value, I2C register reads will be executed as a combined write-read I2C transaction.

- **lck (QNX only)**

By default, the I2C master will be locked during an Apple authentication command. If this causes problems on a target system, the locking can be disabled by setting lck to zero.

A valid option string might look like this:

```
?i2c_delay_microsecs=2000&i2c_write_extra_microsecs=0&i2c_retry_count=200
```

or for QNX systems with a 40 kHz bus access:

```
?i2c_delay_microsecs=2000&i2c_write_extra_microsecs=0&i2c_retry_count=200&sclock=40000
```

HUMAX Confidential

8. Cinemo Events

Cinemo events provide continuous runtime information about changes to the playback state. Applications may receive playback events by [ICinemoPlayer::SetEventCallback\(\)](#) method and specifying an application defined callback function.

It is a design goal of Cinemo Media Engine that the current playback state can be known by monitoring playback events, thus avoiding repeated calls to [ICinemoPlayer::GetStatus\(\)](#).

Note: the event callback may be called by various threads with Cinemo Media Engine. An application which registers for playback events must be prepared to execute code on Cinemo threads, i.e. threads the application did not explicitly create. If this is a problem, a utility interface [ICinemoEventQueue](#) is provided by the Cinemo SDK, which acts as a FIFO between Cinemo threads and the application.

Each event is passed to the application callback function by a CinemoEvent structure, which consists of a unique event code followed by up to seven event specific parameters:

```
typedef struct {
    uint32 code;           /* event code */
    uint32 d[7];           /* event arguments */
    uint32 source_id;      /* event source ID */
} CinemoEvent;
```

- **code**

Event code. See following sections.

- **d[7]**

Event arguments. See following sections.

- **source_id**

User-defined number set by SetEventSourceID() identifying the sender of event.

The following sections describe each event code. It may occur that an application receives events which are not listed or documented below. These events can be safely ignored.

8.1 CINEMO_EC_OPEN

CINEMO_EC_OPEN indicates that the opening of a track for playback was completed.

Parameters

- **d[0]** CinemoError code
- **d[1]** track identifier in the playlist (low 32-bits)
- **d[2]** track identifier in the playlist (upper 32-bits)

Description

CINEMO_EC_OPEN is posted whenever a track in a playlist is opened. The error code indicates if the track was opened successfully.

8.2 CINEMO_EC_OPEN_GAPLESS

CINEMO_EC_OPEN_GAPLESS indicates that the gapless opening of a track for playback was completed.

Parameters

- **d[0]** CinemoError code
- **d[1]** track identifier in the playlist (low 32-bits)
- **d[2]** track identifier in the playlist (upper 32-bits)

Description

CINEMO_EC_OPEN_GAPLESS is posted whenever a track in a playlist is opened in gapless mode. This usually is the case for all tracks in a playlist, except the first one. The event is even posted without any previous user action. The error code indicates if the track was opened successfully.

8.3 CINEMO_EC_DOMAIN

CINEMO_EC_DOMAIN indicates that the playback domain has changed.

Parameters

- **d[0]** playback domain

Description

The playback domain can be used to monitor changes in playback state such as entering and leaving menus, primarily intended for DVD playback scenarios.

8.4 CINEMO_EC_CUE

CINEMO_EC_CUE indicates that the cue state has changed.

Parameters

- **d[0]** cue state

Description

CINEMO_EC_CUE is posted when the cue state has changed. A non-zero cue state indicates that the current play speed could not be honored because the playback engine is buffering data. Playback is always paused in this state, although the current play speed may be non-zero.

8.5 CINEMO_EC_PLAYSPEED

CINEMO_EC_PLAYSPEED indicates that the play speed has changed.

Parameters

- **d[0]** play speed in units of 1000

Description

CINEMO_EC_PLAYSPEED is posted in response to [ICinemoPlayer::SetSpeed\(\)](#), or if for any other reason the play speed changes. Negative values are used for reversed direction. Note that even if the play speed is non-zero, playback may still be paused in cue state (see *CINEMO_EC_CUE*).

8.6 CINEMO_EC_TITLE

CINEMO_EC_TITLE indicates that either the current title number has changed, or the total number of titles changed.

Parameters

- **d[0]** current title
- **d[1]** total number of titles

Description

In most playback scenarios the current title and number of titles will be 1, however some media formats may support multi-title playback, e.g. DVD-Video. If current title value is 0 then there is no current title.

For Blu-ray discs additionally to the one-based title numbers there are two special values referring to:

- **CINEMO_TITLE_FIRST_PLAYBACK** the first playback title
- **CINEMO_TITLE_TOP_MENU** the top menu title

8.7 CINEMO_EC CHAPTER

CINEMO_EC CHAPTER indicates that either the current playing chapter changed, or the total number of chapters changed.

Parameters

- **d[0]** current chapter
- **d[1]** total number of chapters

Description

This event is posted in response to various [ICinemoPlayer](#) methods which may change the current chapter, or during playback when the current time position advances to a new chapter, or if for any other reason the current chapter or total number of chapters changed, e.g. when starting playback of a new track.

8.8 CINEMO_EC_ANGLE

CINEMO_EC_ANGLE indicates that either the current angle (video stream) changed, or the total number of angles (video streams) changed.

Parameters

- **d[0]** current angle
- **d[1]** total number of angles

Description

This event is posted in response to [ICinemoPlayer::SelectAngle\(\)](#), or if for any other reason the current angle or total number of angles changed, e.g. when starting playback of a multi-angle DVD-Video title.

8.9 CINEMO_EC_AUDIO

CINEMO_EC_AUDIO indicates that either the current audio stream changed, or the total number of audio streams changed.

Parameters

- **d[0]** current audio stream
- **d[1]** total number of audio streams

Description

This event is posted in response to [ICinemoPlayer::SelectAudio\(\)](#), or if for any other reason the current audio stream or total number of audio streams changed.

8.10 CINEMO_EC_SUBPICTURE

CINEMO_EC_SUBPICTURE indicates that either the current subpicture stream changed, or the total number of subpicture streams changed.

Parameters

- **d[0]** current subpicture stream
- **d[1]** total number of subpicture streams

Description

This event is posted in response to [ICinemoPlayer::SelectSubpicture\(\)](#), or if for any other reason the current subpicture stream or number of subpicture streams changed.

8.11 CINEMO_EC_BUTTON

CINEMO_EC_BUTTON indicates that either the current selected button changed, or the total count of buttons changed.

Parameters

- **d[0]** current selected button
- **d[1]** total number of buttons

Description

This event is posted in response to [ICinemoPlayer::SelectButton\(\)](#), or if for any other reason the selected button or total number of buttons changed, e.g. when starting playback and during navigation of a DVD-Video menu.

8.12 CINEMO_EC_SUBPICTURE_RECT

CINEMO_EC_SUBPICTURE_RECT indicates that the current video rectangle for rendering subpicture has changed.

Parameters

- **d[0]** subpicture rectangle X1
- **d[1]** subpicture rectangle Y1
- **d[2]** subpicture rectangle X2
- **d[3]** subpicture rectangle Y2

Description

This event is used to normalize XY coordinates for button selection during e.g. DVD-Video menu navigation.

8.13 CINEMO_EC_PROHIBITED_UOPS

CINEMO_EC_PROHIBITED_UOPS indicates that the current prohibited user operations have changed.

Parameters

- **d[0]** bitwise logical OR of prohibited user operations

Description

This event may be used by applications to enable or disable GUI controls.

8.14 CINEMO_EC_STILL

CINEMO_EC_STILL indicates that the still mode has changed.

Parameters

- **d[0]** current still mode

Description

A non-zero still mode indicates that playback is frozen for the specified number of seconds. This frequently occurs when playing DVD-Video menus. A value of zero indicates that still mode has ended, and a value of 255 indicates that still mode is infinite.

8.15 CINEMO_EC_ERROR

Parameters

- **d[0]** CinemoError code

Description

In case of [ICinemoPlayer](#) playback will stop after this event is posted.

8.16 CINEMO_EC_WARNING

CINEMO_EC_WARNING indicates that a recoverable error occurred.

Parameters

- **d[0]** CinemoError code

Description

In case of [ICinemoPlayer](#) playback will still continue after this event is posted.

8.17 CINEMO_EC_PARENTAL_LEVEL

CINEMO_EC_PARENTAL_LEVEL indicates a request for a temporary parental level change.

Parameters

- **d[0]** requested parental level

Description

CINEMO_EC_PARENTAL_LEVEL is posted when DVD navigation commands request a temporary parental level change. The parental level is in range 1 to 8. Applications should respond to this event with a call to [ICinemoPlayer::AcceptParentalLevel\(\)](#), for indicating if the temporary parental level change was accepted.

8.18 CINEMO_EC_TIME

CINEMO_EC_TIME indicates a playback time event.

Parameters

- **d[0]** current playback time in data format as defined in CinemoTimeCode
- **d[1]** track duration in data format as defined in CinemoTimeCode
- **d[2]** buffer utilization in milliseconds (estimated)
- **d[3]** total buffer size in milliseconds (estimated)
- **d[4]** past still buffered content in milliseconds (estimated)
- **d[5]** current playback time in milliseconds
- **d[6]** track duration in milliseconds

Description

CINEMO_EC_TIME is posted at intervals configurable with [ICinemoPlayer::SetTimeEventsInterval\(\)](#) method. Note that the event is also posted aperiodically, such as when playback is paused or resumed. Infinite duration is represented as d[1] being 255 hours, 59 minutes, 59 seconds and 99 cents and d[6] being `UINT32_MAX`. An infinite duration is associated with live stream sources. A zero duration is associated with a source for which the actual duration is unknown.

The formats for the date to indicate playback time and track duration are described by the CinemoTimeCode structure:

```
typedef struct {
    uchar hours;
    uchar mins;
    uchar seconds;
    uchar cents;
} CinemoTimeCode;
```

CinemoTimeCode describes the time in hours, minutes, seconds and 1/100 second units.

Sample

```
const CinemoEventTime& time = (
const CinemoEventTime&) event.d[0];

printf("CINEMO_EC_TIME: %02d:%02d:%02d.%02d of %02d:%02d:%02d.%02d\n",
time.position.hours,
time.position.mins,
time.position.seconds,
time.position.cents,
time.duration.hours,
time.duration.mins,
time.duration.seconds,
time.duration.cents);
```

8.19 CINEMO_EC_FINISHED

CINEMO_EC_FINISHED indicates that the end of playback is reached.

Parameters

- None

Description

Playback finishes when the current playing track ends and the current *CINEMO_PLREPEAT* and *CINEMO_PLORDER* settings are such that there is no next track in the playlist.

8.20 CINEMO_EC_METADATA

CINEMO_EC_METADATA indicates that the content of the player's metapool has changed.

Parameters

- None

Description

This event is posted when the content of the Cinemo player's metapool has changed. To read the metadata, an application should call [ICinemoPlayer::InitMetapool\(\)](#) to obtain a new [ICinemoMetapool](#) interface, or call [ICinemoMetapool::Refresh\(\)](#) on an [ICinemoMetapool](#) interface that has already been obtained from the current [ICinemoPlayer](#) instance.

8.21 CINEMO_EC_PLAYLIST_METADATA

CINEMO_EC_PLAYLIST_METADATA indicates that the content of the player's playlist metapool for the current track has changed.

Parameters

- None

Description

This event is posted when the content of the Cinemo player's playlist metapool has changed because the current track changed, or the underlying playlist's metadata for the current track changed. To read the playlist metadata for the current track, an application should call [ICinemoPlayer::InitPlaylistMetapool\(\)](#) to obtain a new [ICinemoMetapool](#) interface, or call [ICinemoMetapool::Refresh\(\)](#) on an [ICinemoMetapool](#) interface that has already been obtained from the current [ICinemoPlayer](#) instance.

8.22 CINEMO_EC_PLAYLIST_CHANGED

CINEMO_EC_PLAYLIST_CHANGED indicates that the content of the playlist has been updated.

Parameters

- **d[0]** start offset for contiguous tracks for which metadata has changed
- **d[1]** end offset (inclusive) for contiguous tracks for which metadata has changed
- **d[2]** total count of tracks in the playlist

Description

This event is posted from [ICinemoPlaylist](#) if the playlist is created using the [ICinemoPlaylist::Open\(\)](#) method, whenever new information about the tracks becomes available. For example, when opening UPnP playlists, *CINEMO_EC_PLAYLIST_CHANGED* will be posted when the number of tracks in the playlist is known. It will be posted again if the contents of the playlist change (e.g. because the remote server signaled an update event).

It will also be posted when new metadata is fetched for tracks in response to [ICinemoPlaylist::Open\(\)](#), in which case d[0] and d[1] specify the start and end offsets (inclusive) for contiguous tracks for which metadata has become available. When handling *CINEMO_EC_PLAYLIST_CHANGED* events from [ICinemoPlaylist](#), an application may determine which tracks have changed by calling [ICinemoPlaylist::GetIndexOfTrackOffset\(\)](#) for each offset between, and including, the start and end positions indicated in the event. Please see the function description for [ICinemoPlaylist::GetIndexOfTrackOffset\(\)](#) for more information.

8.23 CINEMO_EC_SESSION_DATA

CINEMO_EC_SESSION_DATA indicates that the current distributed playback session pool has changed.

Parameters

- None

Description

CINEMO_EC_SESSION_DATA is posted whenever the distributed playback session pool changes. This will occur after calling [ICinemoPlayer::SetSessionData\(\)](#) on the distributed playback server. The session pool is distributed to all connected clients, and the event is posted simultaneously on the server and all connected clients.

To read the session pool, call [ICinemoPlayer::GetSessionPool\(\)](#). The server and all connected clients will return an identical session pool.

8.24 CINEMO_EC_GRAPH_STATUS

CINEMO_EC_GRAPH_STATUS indicates that the graph status has changed.

Parameters

- **d[0]** audio queued bytes
- **d[1]** video queued bytes
- **d[2]** total queued bytes

Description

CINEMO_EC_GRAPH_STATUS is posted to indicate the graph status changed. It is sent at a minimum at a configurable interval. [ICinemoPlayer::GetGraphStatus\(\)](#) can also be called to get the new graph status.

8.25 CINEMO_EC_VIDEO_STATUS

CINEMO_EC_VIDEO_STATUS indicates that the video status has changed.

Parameters

- **d[0]** set to 1 when there is a video stream, 0 otherwise
- **d[1]** video frame width in pixels
- **d[2]** video frame height in pixels
- **d[3]** video aspect ratio X
- **d[4]** video aspect ratio Y
- **d[5]** device number (zero for primary)

Description

This event is posted by the video renderer, and indicates that either the first decoded video frame is available, or the video resolution has changed.

8.26 CINEMO_EC_AUDIO_STATUS

CINEMO_EC_AUDIO_STATUS indicates that audio status has changed.

Parameters

- **d[0]** set to 1 when there is an audio stream playing, 0 otherwise

Description

This is event is posted by the audio renderer, and indicates that audio playback is starting ($d[0]=1$). $d[0]=0$ is currently not used. For future compatibility applications should check the event parameter. For compatibility with future extensions all other event parameters are set to 0.

8.27 CINEMO_EC_TRACK

CINEMO_EC_TRACK indicates that the current track ID has changed.

Parameters

- **d[0]** current track ID (low 32-bits)
- **d[1]** current track ID (upper 32-bits)

Description

This event is posted when playback advances from one track in the playlist to another.

8.28 CINEMO_EC_CLOSE

CINEMO_EC_CLOSE indicates that the closing of a track or playlist was completed.

Parameters

- **d[0]** CinemoError code

Description

CINEMO_EC_CLOSE is posted in response to [ICinemoPlayer::CloseTrack\(\)](#).

8.29 CINEMO_EC_STATUS

CINEMO_EC_STATUS indicates that the status of a track has changed.

Parameters

- None

Description

CINEMO_EC_STATUS is posted to convey a change in a track's status, such as opening, closing or waiting for another operation.

8.30 CINEMO_EC_POPUP_MENU

CINEMO_EC_POPUP_MENU indicates the popup menu state has changed.

Parameters

- **d[0]** popup menu state

Description

CINEMO_EC_POPUP_MENU is posted when Blu-ray popup menu is shown or hidden due to user interactive key press or triggered by the disc internally.

8.31 CINEMO_EC_SECONDARY_VIDEO

CINEMO_EC_SECONDARY_VIDEO indicates that either the current secondary video stream changed, or the total number of secondary video streams changed.

Parameters

- **d[0]** current secondary video stream
- **d[1]** total number of secondary video streams

Description

This event is posted in response to [ICinemoPlayer2:SelectSecondaryVideo\(\)](#), or if for any other reason the current secondary video stream or total number of secondary video streams changed.

8.32 CINEMO_EC_SECONDARY_AUDIO

CINEMO_EC_SECONDARY_AUDIO indicates that either the current secondary audio stream changed, or the total number of secondary audio streams changed.

Parameters

- **d[0]** current secondary audio stream
- **d[1]** total number of secondary audio streams

Description

This event is posted in response to [ICinemoPlayer2::SelectSecondaryAudio\(\)](#), or if for any other reason the current secondary audio stream or total number of secondary audio streams changed.

8.33 CINEMO_EC_SUBPICTURE_STYLE

CINEMO_EC_SUBPICTURE_STYLE indicates that either the current subpicture style changed, or the total number of subpicture styles changed.

Parameters

- **d[0]** current subpicture style
- **d[1]** total number of subpicture styles

Description

This event is posted in response to [ICinemoPlayer2::SelectSubpictureStyle\(\)](#), or if for any other reason the current subpicture style or total number of subpicture styles changed.

8.34 CINEMO_EC_AUDIO_PROP_CHANGE

CINEMO_EC_AUDIO_PROP_CHANGE indicates that the current audio properties changed.

Description

This event is posted in response to [ICinemoPlayer2::SetAudioParams\(\)](#), or if for any other reason the current audio properties changed.

8.35 CINEMO_EC_VIDEO_PROP_CHANGE

CINEMO_EC_VIDEO_PROP_CHANGE indicates that the current video properties changed.

Description

This event is posted in response to [ICinemoPlayer2::SetVideoParams\(\)](#), or if for any other reason the current video properties changed.

8.36 CINEMO_EC_TITLE_TRANSITION

CINEMO_EC_TITLE_TRANSITION indicates the Blu-ray title transition state has changed.

Parameters

- **d[0]** title transition state

Description

CINEMO_EC_TITLE_TRANSITION is posted when Blu-ray title transition is started or ended.

8.37 CINEMO_EC_CMI

CINEMO_EC_CMI indicates that the content management information (CMI) for the currently rendered content has changed.

Parameters

- **d[0]** Updated content management information as defined in CINEMO_CMI.

Description

CINEMO_EC_CMI is posted whenever the content management information (CMI) associated with the rendered content changes. CMI information is extracted from CSS protected DVD-Video or can be specified manually for a track using CINEMO_METANAME_PROTECTION_FLAGS.

The enum CINEMO_CMI defines the CMI flags that might be set in the event parameter:

```
typedef enum
{
    CINEMO_CMI_PROTECTED,
    CINEMO_CMI_USAGERULE_ICT,           /* image constraint token */
    CINEMO_CMI_USAGERULE_AST,          /* analog sunset token */
    CINEMO_CMI_USAGERULE_DOT,          /* digital only token */

    CINEMO_CMI_USAGERULE_APS_OFF,      /* macrovision analog protection system */
    CINEMO_CMI_USAGERULE_APS_TYPE1,
    CINEMO_CMI_USAGERULE_APS_TYPE2,
    CINEMO_CMI_USAGERULE_APS_TYPE3,

    CINEMO_CMI_CCI_COPYFREE,           /* copy control information */
    CINEMO_CMI_CCI_NOMORECOPIES,
    CINEMO_CMI_CCI_COPYONEGEN,
    CINEMO_CMI_CCI_COPYNEVER,

    CINEMO_CMI_RETENTION_MOVE_MODE,

    CINEMO_CMI_RETENTION_STATE_FOREVER,
    CINEMO_CMI_RETENTION_STATE_ONE_WEEK,
    CINEMO_CMI_RETENTION_STATE_TWO_DAYS,
    CINEMO_CMI_RETENTION_STATE_ONE_DAY,
    CINEMO_CMI_RETENTION_STATE_TWELVE_HOURS,
    CINEMO_CMI_RETENTION_STATE_SIX_HOURS,
    CINEMO_CMI_RETENTION_STATE_THREE_HOURS,
    CINEMO_CMI_RETENTION_STATE_NINETY_MINUTES,

    CINEMO_CMI_EPN_UNASSERTED,
} CINEMO_CMI;
```

If the option CINEMO_OPTION_PROTECTION_ENABLE_ACCEPT_CMI is enabled, the rendering of the content is disabled until a call to [ICinemoPlayer::AcceptCMI\(d\[0\]\)](#) is performed.

Note that the [CINEMO_EC_CMI](#) event is issued when the first sample of protected content enters the rendering graph. In the rendering graph, samples are buffered to allow decoding and rendering of the next frame. Thus, the event arrives slightly prior to the actual rendering.

Sample

```

uint32 cmi = event.d[0];

if(cmi & CINEMO_CMI_PROTECTED)
{
    printf("The content is protected with the following tokens: ");
    if(cmi & CINEMO_CMI_USAGERULE_ICT) printf("ICT ");
    if(cmi & CINEMO_CMI_USAGERULE_AST) printf("AST ");
    if(cmi & CINEMO_CMI_USAGERULE_DOT) printf("DOT ");
    printf("\n");

    printf("Required APS type for the content is: ");
    if(cmi & CINEMO_CMI_USAGERULE_APS_OFF) printf("NONE\n");
    else if(cmi & CINEMO_CMI_USAGERULE_APS_TYPE1) printf("TYPE1\n");
    else if(cmi & CINEMO_CMI_USAGERULE_APS_TYPE2) printf("TYPE2\n");
    else if(cmi & CINEMO_CMI_USAGERULE_APS_TYPE3) printf("TYPE3\n");

    printf("The copy control information for the content is: ");
    if(cmi & CINEMO_CMI_CCI_COPYFREE) printf("COPY FREE\n");
    else if(cmi & CINEMO_CMI_CCI_NOMORECOPIES) printf("NO MORE COPIES\n");
    else if(cmi & CINEMO_CMI_CCI_COPYONEGEN) {
        printf("COPY ONE GENERATION\n");
        if ((cmi & CINEMO_CMI_RETENTION_MOVE_MODE) == 0) {
            printf("MOVE MODE\n");
        }
    }
    else if(cmi & CINEMO_CMI_CCI_COPYNEVER) {
        printf("COPY NEVER\n");
        if ((cmi & CINEMO_CMI_RETENTION_MOVE_MODE) == 0) {
            printf("RETENTION STATE: ");
            switch (cmi & CINEMO_CMI_RETENTION_STATE_MASK) {
                case CINEMO_CMI_RETENTION_STATE_FOREVER:
                    printf("FOREVER\n"); break;
                case CINEMO_CMI_RETENTION_STATE_ONE_WEEK:
                    printf("ONE WEEK\n"); break;
                case CINEMO_CMI_RETENTION_STATE_TWO_DAYS:
                    printf("TWO DAYS\n"); break;
                case CINEMO_CMI_RETENTION_STATE_ONE_DAY:
                    printf("ONE DAY\n"); break;
                case CINEMO_CMI_RETENTION_STATE_TWELVE_HOURS:
                    printf("TWELVE HOURS\n"); break;
                case CINEMO_CMI_RETENTION_STATE_SIX_HOURS:
                    printf("SIX HOURS\n"); break;
                case CINEMO_CMI_RETENTION_STATE_THREE_HOURS:
                    printf("THREE HOURS\n"); break;
                case CINEMO_CMI_RETENTION_STATE_NINETY_MINUTS:
                    printf("NINETY MINUTES\n"); break;
            }
        }
    }
}

```

```
        }
    }
} else if(cmi & CINEMO_CMI_EPN_UNASSERTED)
{
    printf("The content is not protected and EPN is not asserted\n");
} else {
    printf("The content is not protected\n");
}
```

8.38 CINEMO_EC_EOF

CINEMO_EC_EOF indicates that “end-of-file” has been reached.

Parameters

- None

Description

This event is posted during playback when the navigation has reached the end of the file.

8.39 CINEMO_EC_OPERATION_ERROR

CINEMO_EC_OPERATION_ERROR indicates non critical error result of the asynchronous operations of the [ICinemoPlayer2](#) interface.

Parameters

- **d[0]** Cinemo Error Code of the operation.
- **d[1]** operation that reports the error
- **d[2]** argument value of the operation (if any, otherwise 0)

Sometimes the content interaction operations of the [ICinemoPlayer2](#) interface do not achieve the desired effect. In Blu-ray playback key events could be masked, some operations are currently not permitted, etc.

In **d[0]** the type of error is reported as a Cinemo Error Code value. Possible values are:

- **CinemoErrorOperationMasked**
- **CinemoErrorOperationFailed**
- **CinemoErrorOperationProhibited**

In **d[1]** a value from the *CINEMO_OPERATION* enumeration describes as good as possible which operation failed.

Unfortunately (due to the asynchronous nature of the event handling inside the Blu-ray discs application) it is not possible to uniquely identify which [ICinemoPlayer2](#) operation went wrong. Therefore the *CINEMO_OPERATION* enum specifies groups of operations to describe as closely as possible the operation that caused the error response. For some of the operations there is a 1:1 mapping to the corresponding *CINEMO_OPERATION* enum value. On the other hand a single *CINEMO_OPERATION* enum value represents many [ICinemoPlayer2](#) operations. This is especially true for the PostKeyEvent operation, where it is not possible to detect which particular key event failed, if more than one was sent.

In **d[2]** the argument of the failed operation is given. This value is set only when in **d[0]** **CinemoErrorOperationMasked** is specified. In all other cases the value of this field is 0.

This is non critical information for the user. The user of this interface is being informed that he might not have succeeded to achieve his intention. No assumption about the state of the playback should be made depending on the values reported via this event. Some kind of non intrusive visual or acoustic feedback could be generated according to the occurrence of this event.

8.40 CINEMO_EC_PLAYLIST_ORDER

CINEMO_EC_PLAYLIST_ORDER indicates that the playlist order has changed. The event is posted after [ICinemoPlaylist::SetOrder\(\)](#) has been successfully called or the settings on remote devices are changed.

Parameters

- **d[0]** CINEMO_PLORDER order

8.41 CINEMO_EC_PLAYLIST_REPEAT

CINEMO_EC_PLAYLIST_REPEAT indicates that the playlist's repeat mode has changed. The event is posted after [ICinemoPlaylist::SetRepeat\(\)](#) has been successfully called or the settings on remote devices are changed.

Parameters

- **d[0]** CINEMO_PLREPEAT repeat mode

8.42 CINEMO_EC_PREGAP

CINEMO_EC_PREGAP indicates that the pregap status of a currently playing digital audio CD (CDDA) track has changed. A pregap status of one indicates that the currently played audio corresponds to an extended pregap (index 0) of the CDDA. Note that extended pregaps are appended to the track preceding the pregap. In order to read pregap information from the CD, the option *CINEMO_OPTION_CD_PLAYBACK_READ_SUBCHANNEL* needs to be enabled.

Parameters

- **d[0]** uint32 pregap status

8.43 CINEMO_EC_SELECT

CINEMO_EC_SELECT indicates that the [ICinemoPlaylist::Select\(\)](#) or [ICinemoPlaylist::SelectTrack\(\)](#) call has finished.

Parameters

- **d[0]** CinemoError code

8.44 CINEMO_EC_AUDIO_WATERMARK_MUTE

CINEMO_EC_AUDIO_WATERMARK_MUTE indicates that the audio has been muted or turned on again after a mute – due to the recognition of an audio watermark. This event does only occur if an audio watermark detector has been added to Cinemo.

Parameters

- **d[0]** 1 if the audio has been muted, 0 if the audio is on or the audio has been turned on after a mute

8.45 CINEMO_EC_KEYFRAME_REQUEST

CINEMO_EC_KEYFRAME_REQUEST indicates a request of Cinemo to produce a video keyframe on the livestream sender side. This event is currently only used in the context of Miracast livestreams. It is sent during bad reception situations.

Parameters

- None

8.46 CINEMO_EC_PLAYLIST_EMPTY

CINEMO_EC_PLAYLIST_EMPTY indicates that the underlying playlist is empty.

Parameters

- **d[0]** 1 if true, 0 if false

8.47 CINEMO_EC_REMOTE_VOLUME

CINEMO_EC_REMOTE_VOLUME indicates that the volume on remote device has been changed.

Parameters

- **d[0]** integer, the value should be between 0 and 127 inclusively.

8.48 CINEMO_EC_CORRELATION

CINEMO_EC_CORRELATION indicates that the correlator has finished synchronizing the player clocks.

Parameters

- **d[0]** CinemoError code
- **d[1]** Offset w.r.t. the other player used for correlation, in milliseconds.
- **d[2]** Volume factor w.r.t. the other player used for correlation, in units of 1000.

8.49 CINEMO_EC_VIDEO_VIEWPORT

CINEMO_EC_VIDEO_VIEWPORT indicates that the video viewport has changed.

Description

This event is posted by the video renderer, and indicates that the viewport area or zoom factor has changed.

The formats for the video viewport event is described by the `CinemoEventVideoViewport` structure:

```
typedef struct CinemoEventVideoViewport {
    struct {
        sint16 left;
        sint16 top;
        sint16 right;
        sint16 bottom;
    } source;           /* actual source area of original decoded video */
    struct {
        sint16 left;
        sint16 top;
        sint16 right;
        sint16 bottom;
    } destination;      /* actual destination rectangle of scaled video clipped
                           to target area */
    struct {
        sint16 left;
        sint16 top;
        sint16 right;
        sint16 bottom;
    } target;           /* target area to render video into, set by video parameters */
    uint32 device_number; /* multiple cloned device support (zero for primary) */
} CinemoEventVideoViewport;
```

Sample

```
const CinemoEventVideoViewport& viewport = (
const CinemoEventVideoViewport&) event.d[0];

printf("CINEMO_EC_VIDEO_VIEWPORT: %d:%d:%d:%d %d:%d:%d:%d %d:%d:%d:%d\n",
    viewport.source.left,
    viewport.source.top,
    viewport.source.right,
    viewport.source.bottom,
    viewport.destination.left,
    viewport.destination.top,
    viewport.destination.right,
    viewport.destination.bottom,
    viewport.target.left,
    viewport.target.top,
    viewport.target.right,
    viewport.target.bottom);
```

8.50 CINEMO_EC_WINDOW_RENDERCONFIG

CINEMO_EC_WINDOW_RENDERCONFIG indicates that the bounding box of the actually rendered area in the window has changed.

Parameters

- **d[0]** CinemoEventRenderConfig containing the updated bounding box area.

Description

The format of the window renderconfig event type is defined by the CinemoEventRenderConfig structure,

```
typedef struct CinemoEventRenderConfig {
    struct {
        sint16 left;
        sint16 top;
        sint16 right;
        sint16 bottom;
    } area;           /* bounding box area of visible layers */
} CinemoEventRenderConfig;
```

8.51 CINEMO_EC_REPEAT CHAPTER

CINEMO_EC_REPEAT CHAPTER indicates that repetition mode for the chapter has been changed.

Parameters

- **d[0]** 1 if repetition has been enabled, 0 if repetition has been disabled

8.52 CINEMO_EC_REPEAT_TITLE

CINEMO_EC_REPEAT_TITLE indicates that repetition mode for the title has been changed.

Parameters

- **d[0]** 1 if repetition has been enabled, 0 if repetition has been disabled

8.53 CINEMO_EC_HTTP

CINEMO_EC_HTTP is an informational only event provided by the CinemaPlayer for some HTTP sources. Any information here cannot be used directly for error handling. The information might instead be used for logging purposes or deduction of potential causes for a received error.

There is no direct link between these events and a specific URL. HTTP errors can also happen after redirection or during download or streaming of dependent sources.

Parameters

- **d[0]** CinemoError code produced by a current HTTP source
- **d[1]** HTTP connection state. One of *CINEMO_HTTP_STATE_...* values
- **d[2]** An HTTP response status code if available

8.54 CINEMO_EC_INTERNAL

Any event codes greater than or equal to *CINEMO_EC_INTERNAL* should be ignored.

Parameters

- None

9. Cinemo Error Codes

Most Cinemo API functions return a `CinemoError` code. A large number of error codes are defined; many are intended for internal use and are not normally returned from an API function. To convert from an error code into its string equivalent, use the `CinemoErrorToString` method.

Syntax

```
const char * CinemoErrorToString(  
    [in]     CinemoError hr  
) ;
```

Return value

This method returns the string equivalent of the error code. If the string equivalent for a given error code cannot be found, this method will return "CinemoUnknownError".

9.1 Success Codes

The following error codes indicate that an operation succeeded.

- **CinemoNoError**

The operation succeeded.

- **CinemoFalse**

The operation succeeded. Used infrequently for methods which need to return either TRUE, FALSE or FAILURE, this error code indicates FALSE.

9.2 Failure Codes

The following error codes indicate that an operation failed.

- **CinemoErrorFailed**

Generic failure code, the reason is not given.

- **CinemoErrorArguments**

An invalid pointer was passed to a function, or invalid arguments were passed to a function.

- **CinemoErrorResources**

The operation could not be completed because resources (typically but not always system memory) could not be allocated.

- **CinemoErrorState**

The operation is not valid in the current state.

- **CinemoErrorCancel**

The operation was canceled by the user.

- **CinemoErrorEventSignalled**

The operation could not be completed because an event was signaled. In this case the event should be handled before the operation is attempted again.

- **CinemoErrorNotImplemented**

The function is not implemented.

- **CinemoErrorUnexpected**

An unexpected error occurred; this indicates an invalid code path.

- **CinemoErrorImpossible**

An impossible error occurred. Usually, this cannot happen.

- **CinemoErrorDeleted**

The operation could not be completed because the object was deleted. Similar in meaning to *CinemoErrorState*.

- **CinemoErrorMediaType**

The media format was unrecognized. This error code is returned if Cinemo is able to detect the media / file, but the actual media format is not supported.

- **CinemoErrorUnknownStreams**

The media format was recognized, but no valid streams were found.

- **CinemoErrorDevctl**

An invalid or unknown NMECTL was received by the io-nme resource manager.

- **CinemoErrorInterface**

A plugin interface could not be created. Please ensure that SetPluginDirectory() has been called. This error code is also returned if a media / file which is not supported by Cinemo is opened.

- **CinemoErrorTimeout**

The operation could not be completed because a time-out occurred.

- **CinemoErrorPending**

The operation is currently pending.

- **CinemoErrorBusy**

The device is busy.

- **CinemoErrorNotSupported**

The called API function is not supported in this context.

- **CinemoErrorNotConnected**

The operation could not be completed because a required filter pin is not connected.

- **CinemoErrorFlush**

The operation could not be completed because the graph is flushing the stream.

- **CinemoErrorEOS**

The end of stream was reached.

- **CinemoErrorUnderflow**

An underflow occurred.

- **CinemoErrorOverflow**

An overflow occurred.

- **CinemoErrorDrop**

The operation could not be completed because samples were dropped.

- **CinemoErrorFrameStep**

The graph should be resumed from pause state in order to handle the frame step function. This is an internal error code and is not returned by any API function.

- **CinemoErrorFrameSecondField**

Indicates that a second video field is required. This is an internal error code and is not returned by any API function.

- **CinemoErrorDecode**

The stream could not be decoded, indicates a corrupt stream.

- **CinemoErrorRequireMoreData**

CinemoErrorResend

More data is required before the stream can be identified or processed by the function.

- **CinemoErrorScanTime**

CinemoErrorServiceChange

CinemoErrorDiscontinuous

Internal streaming errors, indicating the end of either trick play or graph scan time. These error codes are never returned by external API functions.

- **CinemoErrorEOF**

The end of file was reached.

- **CinemoErrorFileOpen**

The file or path could not be opened.

- **CinemoErrorFileDoesNotExist**

CinemoErrorFileExists

The operation could not be completed because the file does not exist, or because the file already exists.

- **CinemoErrorFileSeek**

The file seek operation failed.

- **CinemoErrorFileRead**

The file could not be read.

- **CinemoErrorFileWrite**

The file could not be written.

- **CinemoErrorCRC**

The file or disc could not be read because a CRC mismatch error occurred.

- **CinemoErrorDiscEjected**

The disc could not be opened because it was ejected, or there is no media in the drive.

- **CinemoErrorDiscTimeout**

The operation could not be completed because timeout occurred.

- **CinemoErrorDiscRead**

The disc could not be read.

- **CinemoErrorDiscSeek**

The disc seek operation failed.

- **CinemoErrorDiscAuthentication**

The DVD disc failed to authenticate.

- **CinemoErrorDiscKey**

The DVD disc failed in the key exchange process.

- **CinemoErrorDiscTOC**

The TOC could not be read from the disc.

- **CinemoErrorDiscCommand**

The operation could not be completed because a generic ATAPI command failure occurred.

- **CinemoErrorSockInUse**

The socket address/port is already in use.

- **CinemoErrorNotEnoughSpace**

Not enough space on disk.

- **CinemoErrorDiscProfile**

Disc profile is not allowed for that media type (example BD9 and BD5 as Blu-ray disc)

- **CinemoErrorUDF**

The UDF file structure is invalid.

- **CinemoErrorDVD**

The DVD navigation structure is invalid.

- **CinemoErrorVMG**

- CinemoErrorVTS**

The DVD VMGI or VTSI is invalid.

- **CinemoErrorRegion**

The DVD could not be opened because of a region mismatch.

- **CinemoErrorDomain**

The navigation function could not be completed because it is not appropriate for the current domain.

- **CinemoErrorStreamUnavailable**

The stream identifier is not currently available.

- **CinemoErrorDRM**

The operation could not be completed because a file is protected by DRM.

- **CinemoErrorProhibitedUOP**

The operation could not be completed because it is prohibited by the disc manufacturers.

- **CinemoErrorParentalLevel**

The operation could not be completed because the parental level is not set to high enough value.

- **CinemoErrorAppRunning**

The closing of Blu-ray disc has finished, but there are still disc-unbound applications running in background.

- **CinemoErrorOperationMasked**

- CinemoErrorOperationFailed**

The operation was masked or failed inside the content code. These errors apply only for Blu-ray playback, where the on disc content code, which is executed in the Cinemo Media Engine returned corresponding error results.

- **CinemoErrorOperationProhibited**

The operation could not be completed because it is prohibited in the current context.

- **CinemoErrorVideoResources**

The operation could not be completed because video hardware resources could not be allocated. Similar to *CinemoErrorResources*.

- **CinemoErrorVideoClosed**

- CinemoErrorSeeking**

- CinemoErrorGapless**

Internal error codes for special use. These should never be returned by any Cinemo API.

- **CinemoErrorKeyNotUnique**

Key not unique.

- **CinemoErrorDisabled**

Disabled feature.

- **CinemoErrorFormatChanged**

Format change between currently played file and new attached one.

- **CinemoErrorMaxH264Level**

Attached video is encoded with higher H264 level than allowed to play.

- **CinemoErrorMaxResolution**

Attached video has higher resolution than allowed to play.

- **CinemoErrorMaxVxdFileMemory**

Attached video needs more memory to decode than allowed to use.

- **CinemoErrorMaxVxdTotalMemory**

All files in sum need more memory to decode than allowed to use.

- **CinemoErrorRedirection**

Attached URL points to another URL. This is not allowed by definition.

- **CinemoErrorConnectionRefused**

Server which provides the attached URL refused the connection (HTTP error code >= 400).

- **CinemoErrorConnectionFailed**

Connection to given URL failed. Several reasons: network error, firewall misconfiguration, etc.

- **CinemoErrorSourcePort**

Source port range is exhausted; no free source ports (in the configured range) are available.

- **CinemoErrorLosslessAudioCodec**

Playback of DTS-HD MA is not allowed (can be changed in configuration).

- **CinemoErrorRange**

HTTP Range request is invalid.

- **CinemoErrorDNSLookup**

Requested URL could not be found.

- **CinemoErrorNoSuchObject**

Requested metadata item, Media Management node or container could not be found.

- **CinemoErrorNoMoreItems**

Navigation has reached the end of the playlist for forward searches, or the beginning of the playlist for backward searches.

- **CinemoErrorSampleRate**

An audio track has been rejected due to invalid sample rate, or the requested sample rate to be used for sample rate conversion is invalid.

- **CinemoErrorOutputCompliance**

Checking of protected output compliance detected a violation of compliance rules during playback of protected content (e.g. Blu-ray).

- **CinemoErrorKeyRevoked**

A cryptographic key used by the system to ensure cryptographic integrity of the content (e.g. Blu-ray) or a connection (e.g. DTCP) has been revoked.

- **CinemoErrorBadCertificate**

A cryptographic certificate used by the system e.g. for DTCP could not be verified successfully.

- **CinemoErrorContentVerification**

Cryptographic verification of content (e.g. BD-Live content) against its stated issuer failed.

- **CinemoErrorContentDecryption**

Decryption of DRM protected content failed.

- **CinemoErrorMediaNotProtected**

Content is not DRM protected, but it is expected to be protected in order to fulfill DRM compliance rules.

- **CinemoErrorContentProtectionInit**

The initialization phase of a cryptographic content protection system failed (e.g. title startup on BD+ protected Blu-ray discs).

- **CinemoErrorContentPermission**

Content does not provide enough or correct permissions to be decrypted for playback (e.g. Blu-ray).

- **CinemoErrorSecondaryVideo**

Attempt to playback Blu-ray PiP secondary video, but option *CINEMO_OPTION_BD_SECONDARY_VIDEO* is disabled).

- **CinemoErrorUnsupportedAudio**

Attempt to open a file with audio track(s), but none of them being supported (option *CINEMO_OPTION_FILE_FAIL_UNSUPPORTED_AUDIO* is enabled).

- **CinemoErrorUnsupportedVideo**

Attempt to open a file with video track(s), but none of them being supported (option *CINEMO_OPTION_FILE_FAIL_UNSUPPORTED_VIDEO* is enabled).

- **CinemoErrorAudioWatermarkStop**

The audio watermark detector has stopped the playback due to the recognition of a watermark. This error code can only occur if an audio watermark detector has been added to Cinemo.

- **CinemoErrorNoSuchContainer**

Attempt to access media management with an invalid pid.

- **CinemoErrorVolumeMounted**

Media management request not possible because volume is mounted.

- **CinemoErrorVolumeDismounted**

Media management request not possible because volume is dismounted.

- **CinemoErrorSearchCriteria**

Invalid media management search criteria.

- **CinemoErrorSortCriteria**

Invalid media management sort criteria.

- **CinemoErrorAuthDevice**

Failure when accessing authentication device.

- **CinemoErrorTrackChanged**

The current playing track on a remote device has changed.

- **CinemoErrorConfiguration**

The requested configuration is invalid.

- **CinemoErrorMaxInstances**

The maximum number of instances for a specific resource is already in use and another instance could not be created. Currently this applies to the number of allowed Dolby decoder instances of a Cinemo Dolby decoder component as defined by the component license.

- **CinemoErrorDataIntegrity**

The data integrity has been compromised.

- **CinemoErrorDeviceLifecycle**

The device cannot be opened because the lifecycle is not properly defined. Either no lifecycle object has been opened yet or an additional lifecycle object is being created.

- **CinemoErrorDeviceOccupied**

An underlying device can not fulfill the request because other functionality used on the device prevents it.

- **CinemoErrorDeviceSupportMissing**

The underlying device is not designed for this kind of functionality.

- **CinemoErrorDeviceFunctionDisabled**

The underlying device may support the functionality, but it depends on conditions out of Cinemo's control to provide the function for this device (e.g. the device signaled unavailability of the function).

- **CinemoErrorDeviceStunned**

The underlying device suddenly stopped to react in normal way. This might be because a (re)boot of the device, a language change on the device or some other temporary hiccup of the device is in progress. This error code is returned only if this situation is detectable. Usually an attempt to reconnect the device (optionally after resetting the link to the device) will be successful again.

- **CinemoErrorDeviceAuthentication**

The authentication with a connected device has failed.

- **CinemoErrorDriverFailure**

The device driver required for the requested action has failed.

- **CinemoErrorDeviceState**

The remote device has some inconsistent state.

- **CinemoErrorFeatureNotLicensed**

A feature which has not been licensed with Cinemo has been requested.

- **CinemoErrorTargetPlatform**

Cinemo is not allowed to run on the target platform.

- **CinemoErrorFeatureNotInSKU**

The use of a feature was requested which is not available in the configured SKU.

- **CinemoErrorTopology**

In the context of HDCP, the intended topology is invalid.

- **CinemoErrorMTPProtocol**

There were severe MTP errors when performing MTP operations. These errors include “invalid transaction id”, “USB IO errors” and so on.

- **CinemoErrorMTPSessionNotOpen**

This error is returned in cases where the MTP devices closes the MTP session.

- **CinemoErrorAccountInvalid**

This error is returned in cases of invalid account credentials.

- **CinemoErrorAccountInactive**

This error is returned in cases of a valid account, which is not accessible.

- **CinemoErrorAccountPremiumRequired**

This error is returned in cases where an account needs to be upgraded to be usable.

10. Cinemo Media Types

The following sections list the media types and subtypes supported by Cinemo. All information about a media type is encapsulated in the CinemoMediaType structure is defined here:

```
typedef struct {
    CINEMO_MEDIATYPE type;
    CINEMO_MEDIASUBTYPE subtype;
    CinemoMediaFormat format;
    CinemoMediaLang lang;
    CinemoMediaTypeHeader header;
} CinemoMediaType;
```

- **type**

See [Media Types](#)

- **subtype**

See [Media Subtypes](#)

- **format**

See [Media Format](#)

- **lang**

Specifies the language, if known, as ISO 639-1 code. This will be zero if unknown.

- **header**

Not currently used.

Cinemo provides utility functions to convert media types and subtypes into UTF-8 text strings which may be used, for example, to display in an application's GUI.

```
const char * CinemoMediaTypeToString(
    [in] CINEMO_MEDIATYPE type
);
const char * CinemoMediaSubtypeToString(
    [in] CINEMO_MEDIASUBTYPE subtype
);
```

10.1 Media Types

The following media types are defined. Please note that types may be defined but not supported by a particular build of the Cinemo Media Engine.

CINEMO_MEDIATYPE	DESCRIPTION
CINEMO_MEDIATYPE_VIDEO	Video
CINEMO_MEDIATYPE_AUDIO	Audio
CINEMO_MEDIATYPE_SUBPICTURE	Subpicture (except DVD)
CINEMO_MEDIATYPE_DVD	DVD Subpicture (DVD only)
CINEMO_MEDIATYPE_MP4	MPEG 4 encoding. Please note that due to internal details of Cinemo's architecture this mediatype might also be used for files which are not MPEG 4 encoded.
CINEMO_MEDIATYPE_OGG	Not currently used
CINEMO_MEDIATYPE_PES_PAYLOAD	Not currently used
CINEMO_MEDIATYPE_LATM	Low Overhead Audio Transport Multiplex
CINEMO_MEDIATYPE_ADTS	Audio Data Transport Stream encoding
CINEMO_MEDIATYPE_ADIF	Audio Data Interchange Format

10.2 Media Subtypes

The following audio, video and image subtypes are defined. Please note that subtypes may be defined but not supported by a particular build of the Cinemo Media Engine.

10.2.1 Audio Subtypes

CINEMO_MEDIASUBTYPE	DESCRIPTION
CINEMO_MEDIASUBTYPE_MPEG1AudioL1	ISO/IEC 11172-3 MPEG-1 Layer 1
CINEMO_MEDIASUBTYPE_MPEG1AudioL2	ISO/IEC 11172-3 MPEG-1 Layer 2
CINEMO_MEDIASUBTYPE_MPEG1AudioL3	ISO/IEC 11172-3 MPEG-1 Layer 3 ('MP3')
CINEMO_MEDIASUBTYPE_MPEG2AudioL1	ISO/IEC 13818-3 MPEG-2 Layer 1
CINEMO_MEDIASUBTYPE_MPEG2AudioL2	ISO/IEC 13818-3 MPEG-2 Layer 2
CINEMO_MEDIASUBTYPE_MPEG2AudioL3	ISO/IEC 13818-3 MPEG-2 Layer 3 ('MP3')
CINEMO_MEDIASUBTYPE_MPEG2AudioL3Pro	mp3Pro
CINEMO_MEDIASUBTYPE_DOLBY	ETSI TS 102 366 Dolby Digital
CINEMO_MEDIASUBTYPE_DOLBY_SURROUND	Dolby Surround
CINEMO_MEDIASUBTYPE_DOLBY_SURROUND_EX	Dolby Surround EX
CINEMO_MEDIASUBTYPE_DOLBY_HEADPHONE	Dolby Headphone
CINEMO_MEDIASUBTYPE_DOLBY_PLUS	ETSI TS 102 366 Dolby Digital Plus
CINEMO_MEDIASUBTYPE_DOLBY_PLUS_CORE	ETSI TS 102 366 Dolby Digital Plus (Dolby Digital Core)
CINEMO_MEDIASUBTYPE_DTS	DTS Coherent Acoustics
CINEMO_MEDIASUBTYPE_DTS_9624	DTS 96 kHz 24-bit
CINEMO_MEDIASUBTYPE_DTS_ES_MATRIX	DTS ES Matrix
CINEMO_MEDIASUBTYPE_DTS_ES_DISCRETE	DTS ES Discrete
CINEMO_MEDIASUBTYPE_DTS_HD	DTS HD
CINEMO_MEDIASUBTYPE_DTS_HD_MA	DTS HD MA
CINEMO_MEDIASUBTYPE_DTS_LOSSLESS	DTS Lossless Extension Specification 3.2
CINEMO_MEDIASUBTYPE_DTS_LBR	DTS LBR
CINEMO_MEDIASUBTYPE_LPCM_DVD	Linear PCM (DVD)
CINEMO_MEDIASUBTYPE_LPCM_DVDA	Linear PCM (DVD-Audio)
CINEMO_MEDIASUBTYPE_LPCM_HDDVD	Linear PCM (HD DVD)
CINEMO_MEDIASUBTYPE_LPCM_BD	Linear PCM (Blu-ray)
CINEMO_MEDIASUBTYPE_AAC_LC	ISO/IEC 13818-7 MPEG-2 AAC LC ISO/IEC 14496-3 MPEG-4 AAC LC
CINEMO_MEDIASUBTYPE_AAC_LD	ISO/IEC 14496-3 MPEG-4 AAC LD

CINEMO_MEDIASUBTYPE	DESCRIPTION
CINEMO_MEDIASUBTYPE_AAC_MAIN	ISO/IEC 13818-7 MPEG-2 AAC Main ISO/IEC 14496-3 MPEG-4 AAC Main
CINEMO_MEDIASUBTYPE_AAC_SSR	ISO/IEC 13818-7 MPEG-2 AAC SSR ISO/IEC 14496-3 MPEG-4 AAC SSR
CINEMO_MEDIASUBTYPE_AAC_LTP	ISO/IEC 13818-7 MPEG-2 AAC LTP ISO/IEC 14496-3 MPEG-4 AAC LTP
CINEMO_MEDIASUBTYPE_AAC_HE	ISO/IEC 14496-3 MPEG-4 AAC HE (High Efficiency, aacPlus)
CINEMO_MEDIASUBTYPE_AAC_HEV2	ISO/IEC 14496-3 MPEG-4 AAC HE v2 (High Efficiency v2, Enhanced aacPlus)
CINEMO_MEDIASUBTYPE_AAC_BSAC	ISO/IEC 13818-7 MPEG-2 AAC BSAC ISO/IEC 14496-3 MPEG-4 AAC BSAC
CINEMO_MEDIASUBTYPE_AAC_ELD	ISO/IEC 14496-3 MPEG-4 AAC ELD
CINEMO_MEDIASUBTYPE_PCM	PCM
CINEMO_MEDIASUBTYPE_IeeePCM	PCM (IEEE754 floating point format)
CINEMO_MEDIASUBTYPE_ADPCM_Microsoft	ADPCM
CINEMO_MEDIASUBTYPE_ADPCM_IMA_Apple	IMA ADPCM
CINEMO_MEDIASUBTYPE_ADPCM_IMA_Microsoft	Microsoft ADPCM
CINEMO_MEDIASUBTYPE_ADPCM_VOX	Dialogic ADPCM Algorithm 00-1366-001
CINEMO_MEDIASUBTYPE_ADPCM_Creative	ADPCM (Creative)
CINEMO_MEDIASUBTYPE_ADPCM_Flash	ADPCM (Flash)
CINEMO_MEDIASUBTYPE_Alaw	PCM with A-law companding
CINEMO_MEDIASUBTYPE_Mulaw	PCM with μ -law companding
CINEMO_MEDIASUBTYPE_GSM610	3GPP TS 06.10 GSM
CINEMO_MEDIASUBTYPE_MLP	MLP-FBA 1.0, MLP-FBB 1.0
CINEMO_MEDIASUBTYPE_AMR_NB	3GPP TS 26.073
CINEMO_MEDIASUBTYPE_AMR_WB	ITU-T G.722.2
CINEMO_MEDIASUBTYPE_WMA1	Microsoft WMA
CINEMO_MEDIASUBTYPE_WMA2	Microsoft WMA
CINEMO_MEDIASUBTYPE_WMA9_PRO	Microsoft WMA
CINEMO_MEDIASUBTYPE_WMA9_LOSSLESS	Microsoft WMA
CINEMO_MEDIASUBTYPE_WMA9_VOICE	Microsoft WMA
CINEMO_MEDIASUBTYPE_FLAC	FLAC (http://flac.sourceforge.net)
CINEMO_MEDIASUBTYPE_Vorbis	Vorbis I Specification, Feb 2010 (http://www.xiph.org)

CINEMO_MEDIASUBTYPE	DESCRIPTION
CINEMO_MEDIASUBTYPE_SBC	A2DP Specification 12.0 Appendix B
CINEMO_MEDIASUBTYPE_Speex	The Speex Codec Manual Version 1.2 Beta 3 (http://www.xiph.org)
CINEMO_MEDIASUBTYPE_G726	ITU-T G.726
CINEMO_MEDIASUBTYPE_G722_1	ITU-T G.722.1
CINEMO_MEDIASUBTYPE_APE	Monkey's Audio
CINEMO_MEDIASUBTYPE_ALAC	Apple Lossless
CINEMO_MEDIASUBTYPE_RA_144	RealAudio
CINEMO_MEDIASUBTYPE_RA_288	RealAudio
CINEMO_MEDIASUBTYPE_COOK	RealAudio
CINEMO_MEDIASUBTYPE_SI PRO	RealAudio
CINEMO_MEDIASUBTYPE_ATRAC3	RealAudio
CINEMO_MEDIASUBTYPE_RALF	RealAudio
CINEMO_MEDIASUBTYPE_OPUS	Opus
CINEMO_MEDIASUBTYPE_LPCM_DVD_IEEE1394	DVD LPCM over IEEE1394 Bus

10.2.2 Video Subtypes

CINEMO_MEDIASUBTYPE	DESCRIPTION
CINEMO_MEDIASUBTYPE_MPEG1Video	ISO/IEC 11172-2 MPEG-1
CINEMO_MEDIASUBTYPE_MPEG2Video	ISO/IEC 13818-2 MPEG-2
CINEMO_MEDIASUBTYPE_AVC	ISO/IEC 14496-10 MPEG-4 AVC (ITU-T H.264)
CINEMO_MEDIASUBTYPE_HEVC	ISO/IEC 23008-2 MPEG-H Part 2 (ITU-T H.265)
CINEMO_MEDIASUBTYPE_VC1	SMPTE 421M-2006
CINEMO_MEDIASUBTYPE_CVID	Cinepak
CINEMO_MEDIASUBTYPE_CRAM	Microsoft Video 1
CINEMO_MEDIASUBTYPE_DVSD	ISO/IEC 61834-2 DV
CINEMO_MEDIASUBTYPE_MPEG4Video	ISO/IEC 14496-2 MPEG-4 SP/ASP
CINEMO_MEDIASUBTYPE_DIVX	DivX Version 4
CINEMO_MEDIASUBTYPE_FMP4	FFMpeg MPEG-4
CINEMO_MEDIASUBTYPE_MP41	MS MPEG-4 v1
CINEMO_MEDIASUBTYPE_MP42	MS MPEG-4 v2
CINEMO_MEDIASUBTYPE_MP43	MS MPEG-4 v3
CINEMO_MEDIASUBTYPE_FLV1	Adobe Flash MPEG-4
CINEMO_MEDIASUBTYPE_XVID	Xvid

CINEMO_MEDIASUBTYPE	DESCRIPTION
CINEMO_MEDIASUBTYPE_3IVX	3ivx
CINEMO_MEDIASUBTYPE_H263	ITU-T H.263
CINEMO_MEDIASUBTYPE_VP6	On2 VP6
CINEMO_MEDIASUBTYPE_VP8	On2 VP8
CINEMO_MEDIASUBTYPE_WMV7	Microsoft WMV
CINEMO_MEDIASUBTYPE_WMV8	Microsoft WMV
CINEMO_MEDIASUBTYPE_WMV9	Microsoft WMV
CINEMO_MEDIASUBTYPE_Theora	Theora (http://libtheora.org)
CINEMO_MEDIASUBTYPE_MJPEG	MJPEG (QuickTime Format Specification)
CINEMO_MEDIASUBTYPE_RAWVIDEO	Raw Video
CINEMO_MEDIASUBTYPE_RV10	RealVideo
CINEMO_MEDIASUBTYPE_RV20	RealVideo
CINEMO_MEDIASUBTYPE_RV30	RealVideo
CINEMO_MEDIASUBTYPE_RV40	RealVideo
CINEMO_MEDIASUBTYPE_AVS	GB/T 20090.2-2006, GB/T 20090.16-201x
CINEMO_MEDIASUBTYPE_RFB	RFB protocol v3.8 server to client messages

10.2.3 Image Subtypes

CINEMO_MEDIASUBTYPE	DESCRIPTION
CINEMO_MEDIASUBTYPE_JPEG	JPEG
CINEMO_MEDIASUBTYPE_JPEG_BASELINE	Baseline JPEG
CINEMO_MEDIASUBTYPE_JPEG_EXT	JPEG with specification extensions
CINEMO_MEDIASUBTYPE_JPEG_PROGRESSIVE	Progressive JPEG
CINEMO_MEDIASUBTYPE_JPEG_LOSSLESS	Lossless JPEG
CINEMO_MEDIASUBTYPE_PNG	Portable Network Graphics
CINEMO_MEDIASUBTYPE_GIF	Graphics Interchange Format
CINEMO_MEDIASUBTYPE_GIF_87A	GIF version 87a (original)
CINEMO_MEDIASUBTYPE_GIF_89A	GIF version 89a
CINEMO_MEDIASUBTYPE_TIFF	Tagged Image File Format
CINEMO_MEDIASUBTYPE_BMP	Bitmap Image File
CINEMO_MEDIASUBTYPE_WBMP	Wireless Bitmap Image File

10.2.4 Uncompressed Subtypes

CINEMO_MEDIASUBTYPE	DESCRIPTION
CINEMO_MEDIASUBTYPE_ARGB	RGBA color space information with Alpha first
CINEMO_MEDIASUBTYPE_AYUV	YUV color space with combined YUV and alpha
CINEMO_MEDIASUBTYPE_IndexedARGB	ARGB with indexed colors
CINEMO_MEDIASUBTYPE_IndexedAYUV	AYUV with indexed colors
CINEMO_MEDIASUBTYPE_DXVA	DirectX Video Acceleration
CINEMO_MEDIASUBTYPE_EGL	OpenGL
CINEMO_MEDIASUBTYPE_Y800	YUV color space with single Y (monochrome)
CINEMO_MEDIASUBTYPE_NV12	YUV color space with Y and interleaved U/V
CINEMO_MEDIASUBTYPE_NV12_VPU	Same as NV12 using physical memory
CINEMO_MEDIASUBTYPE_YV12	YUV color space with Y and subsampled V/U
CINEMO_MEDIASUBTYPE_I420	YUV color space with Y and subsampled U/V
CINEMO_MEDIASUBTYPE_YUY2	Same as UYVY but different component ordering
CINEMO_MEDIASUBTYPE_UYVY	YUV 4:2:2 color space
CINEMO_MEDIASUBTYPE_PremulARGB	ARGB with premultiplied colors
CINEMO_MEDIASUBTYPE_RGBA	RGBA color space information with Alpha last

10.2.5 Subtitle Subtypes

CINEMO_MEDIASUBTYPE	DESCRIPTION
CINEMO_MEDIASUBTYPE_SubpictureDVD	DVD
CINEMO_MEDIASUBTYPE_SubpictureVCD	Video CD
CINEMO_MEDIASUBTYPE_SubpictureDVB	Digital Video Broadcasting
CINEMO_MEDIASUBTYPE_SubpictureISDB	Integrated Services Digital Broadcasting
CINEMO_MEDIASUBTYPE_SubpictureTTXT	MPEG-4 Streaming text format
CINEMO_MEDIASUBTYPE_SubpictureDxsb	DivX
CINEMO_MEDIASUBTYPE_SubpictureUSF	Universal Subtitle Format
CINEMO_MEDIASUBTYPE_SubpictureSSA	SubStation Alpha
CINEMO_MEDIASUBTYPE_SubpictureSRT	SubRip
CINEMO_MEDIASUBTYPE_SubpictureCheetah	Cheetah
CINEMO_MEDIASUBTYPE_SubpictureUlead	Ulead
CINEMO_MEDIASUBTYPE_SubpictureBD	Blu-ray
CINEMO_MEDIASUBTYPE_SubpictureBD_PIP	Blu-ray picture-in-picture
CINEMO_MEDIASUBTYPE_SubpictureExternal	External subtitles

10.3 Media Format

The CinemoMediaFormat structure is defined here. The format is a union of two possible structures, depending on if the media type describes a video or audio media format.

```
typedef enum {
    CINEMO_FORMATTYPE_NONE = 0,
    CINEMO_FORMATTYPE_UNSPECIFIED = 0,
    CINEMO_FORMATTYPE_VIDEO,
    CINEMO_FORMATTYPE_AUDIO,
} CINEMO_FORMATTYPE;
```

```
typedef struct {
    CINEMO_FORMATTYPE type;
    union {
        CinemaVideoFormat video;
        CinemaAudioFormat audio;
    };
} CinemoMediaFormat;
```

- **type**

Specifies which format structure is used to describe this media type. The format structures for video and audio are defined in the next sections.

- **video**

[Video Format](#), if type is *CINEMO_FORMATTYPE_VIDEO*.

- **audio**

[Audio Format](#), if type is *CINEMO_FORMATTYPE_AUDIO*.

10.3.1 Cinemo Video Format

The CinemoVideoFormat structure is defined here:

```
typedef struct {
    uint32 cx;                                /* frame width */
    uint32 cy;                                /* frame height */
    uint32 frame_duration;                     /* frame duration in 70.56mhz units */
    sint32 naluhadersize;                      /* for MP4 containers */
    CinemoAspectRatio aspect;                  /* display aspect ratio */
    uint32 reserved[2];                         /* padding */
    CinemoVideoFormatFlags flags;              /* flags */
    CinemoRect crop;                           /* video crop rectangle */
    CinemoRect padding;                        /* video padding borders */
    CinemoRect position;                       /* video position (Blu-Ray) */
} CinemoVideoFormat;
```

```
typedef struct {
    uchar interlaced : 1;                      /* interlaced */
    uchar topfieldfirst : 1;                    /* when interlaced, top field is first */
    uchar navigator_aspect : 1;                 /* use container aspect ratio */
    uchar profile_level_valid : 1;              /* profile / level are valid */
    uchar chunked : 1;                          /* packets are chunked */
    uchar decoding_time_stamps : 1;             /* decoding time stamps */
    uchar mixer_coordinates : 1;                /* use mixer coordinate system */
    uchar lumakey_used : 1;                     /* secondary video luma keying */
    uchar display_mode;                         /* CINEMO_VIDEODISPLAYMODE */
    uchar capture_format;                       /* CINEMO_VIDEOCAPTUREFORMAT */
    uchar profile;                            /* profile */
    uchar level;                             /* level */
    uchar stereo_mode;                         /* CINEMO_VIDEOSTEREO MODE */
    uchar view_count;                          /* scene number of views */
    uchar view_index;                          /* scene view index */
    uchar luma_key;                           /* luma key limit */
    uchar pip_scale;                           /* PiP scaling mode (Blu-Ray) */
    uchar yuv_matrix;                          /* YUV transfer matrix */
    uchar yuv_range;                           /* YUV nominal range */
    uchar rotation;                           /* CINEMO_VIDEOROTATION (clockwise) */
    uchar low_latency : 1;                     /* disable frame reordering */
    uchar interlaced_sequence : 1;              /* interlaced sequence */
    uchar viewport_coordinates : 1;            /* use viewport coordinate system */
    uchar still_image;                         /* still image */
    uchar reserved_bits : 4;                   /* padding */
    uchar reserved[2];                         /* ensure 64-bit alignment */
} CinemoVideoFormatFlags;
```

- **cx**

Frame width in pixels including padding. Use **crop** rectangle to query actual video size.

- **cy**

Frame height in pixels including padding. Use **crop** rectangle to query actual video size.

- **frame_duration**

Frame duration in 70.56MHz units.

- **naluheadersize**

Used internally for MP4 containers. Default to zero.

- **aspect**

Frame aspect ratio. Example 4:3 or 16:9. If unknown or not set: 0:0.

- **flags.interlaced**

Video sequence is interlaced.

- **flags.topfieldfirst**

Used internally. Default to zero.

- **flags.navigator_aspect**

Used internally. Default to zero.

- **flags.profile_level_valid**

Video codec profile and level is detected and written to **flags.profile**, **flags.level** fields.

- **flags.chunked**

Used internally. Default to zero.

- **flags.decoding_time_stamps**

Used internally. Default to zero.

- **flags.mixer_coordinates**

Used internally. Default to zero.

- **flags.lumakey_used**

Used internally. Default to zero.

- **flags.display_mode**

DVD display mode. Default to zero.

- **flags.capture_format**

DVD and VCD capture format. Default to zero.

- **flags.profile, flags.level**

Specify video codec profile and level. Valid when **flags.profile_level_valid** is set.

- **flags.stereo_mode**

Used internally. Default to zero.

- **flags.view_count**

Used internally. Default to zero.

- **flags.view_index**

Used internally. Default to zero.

- **flags.luma_key**

Used internally. Default to zero.

- **flags.pip_scale**

Used internally. Default to zero.

- **flags.yuv_matrix**

Used internally. Default to zero.

- **flags.yuv_range**

Used internally. Default to zero.

- **flags.rotation**

Used internally. Default to zero.

- **flags.low_latency**

Used internally. Default to zero.

- **flags.interlaced_sequence**

Used internally. Default to zero.

- **flags.viewport_coordinates**

Used internally. Default to zero.

- **flags.still_image**

Used internally. Default to zero.

- **crop**

Specifies the actual video area should be shown during playback. For example, H.264 1080p video has **cx**=1920, **cy**=1088 frame size, but **crop** is left=0, top=0, right=1920, height=1080.

- **padding**

Used internally. Default to zero.

- **position**

Used internally. Default to zero.

10.3.2 Cinemo Audio Format

The CinemoAudioFormat structure is defined here:

```
typedef struct {
    uint32 samplerate;
    uint32 channels;
    uint32 channelconfig;
    uint32 bits;
    uint32 type;
    uint32 blockalign;
    uint32 byterate;
    uint32 flags;
} CinemoAudioFormat;
```

- **samplerate**

Specify audio sample rate in Hz.

- **channels**

Specify number of channels

- **channelconfig**

Specify channel configuration bitmask. See **CINEMO_CHANNELCONFIG** defines.

- **bits**

Specify number of bits per sample. Zero for non-PCM audio.

- **type**

Specify PCM format type. Zero for non-PCM audio. See **CINEMO_SAMPLETYPE** defines.

- **blockalign**

Specify audio block alignment size. For PCM audio the formula is **channels * bits / 8**

- **byterate**

Specify audio byterate (bitrate / 8).

- **flags**

Additional audio flags.

11. Cinemo Configuration Options

Configuration options are used to control the behavior of Cinemo Media Engine. A default set of options is compiled into the Cinemo binaries for each target platform, and these default values normally guarantee the correct operation of Cinemo on the intended platform.

All options may be set programmatically using the [ICinemoOption](#) interface.

Cinemo demo applications provide a GUI for editing Cinemo configuration options, and will save the options to a `cinemo_options.xml` file in the Cinemo working directory. If present, these options are automatically loaded and applied when starting Cinemo demo applications.

11.1 General Options

CINEMO_OPTION_FILE_TIMEOUT

TYPE	INTEGER
DESCRIPTION	Specify a timeout value, in milliseconds, after which file open operations will fail. This value should in general be large enough to support opening files on CD or DVD which may require longer spin-up times. This option applies to all files except HTTP and FTP (see below).
RANGE	0 - INFINITE. A value of zero will disable the timeout.
DEFAULT VALUE	30000

CINEMO_OPTION_FILE_DUMP_FOLDER

TYPE	STRING
DESCRIPTION	Specify the name of the directory where output dump files can be written. Used for various debugging purposes.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_FILE_FAIL_UNSUPPORTED_VIDEO

TYPE	BOOLEAN
DESCRIPTION	Specify if an attempt to open a source with an unsupported video stream shall fail with <i>CinemoErrorUnsupportedVideo</i> . If disabled with FALSE, such source will be opened with audio streams only (if present).
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_FILE_FAIL_UNSUPPORTED_AUDIO

TYPE	BOOLEAN
DESCRIPTION	Specify if an attempt to open a video source with only unsupported audio streams shall fail with <i>CinemoErrorUnsupportedAudio</i> . If disabled with FALSE, such source will be opened with video stream(s) only.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_FILE_CORRUPTION_PROBABILITY

TYPE	INTEGER
DESCRIPTION	For inserting random corruption when reading files. Specify the number of corrupted bytes inserted per 16MB of data. Used for testing the robustness of Cinemo Media Engine.
RANGE	0 - 100000000. A value of zero will disable random corruption.
DEFAULT VALUE	0

CINEMO_OPTION_FILE_CORRUPTION_SEED

TYPE	INTEGER
DESCRIPTION	For inserting random corruption when reading files. Specify the random number generator seed value. This allows the same corruption to be reproduced on each run.
RANGE	0 - INFINITE
DEFAULT VALUE	1

CINEMO_OPTION_BUFFER_STREAM_KB

TYPE	INTEGER
DESCRIPTION	Specify the size in KB of all playback and streaming buffers. This value should normally only be changed after consultation with Cinemo. For example, certain file or stream types require a minimum size buffer in order to de-multiplex separate audio and video streams contained within the buffer.
RANGE	1 - INFINITE
DEFAULT VALUE	8192

CINEMO_OPTION_BUFFER_STREAM_PREFETCH_KB

TYPE	INTEGER
DESCRIPTION	This option applies to navigators that prefetch live stream data, including DMB and Cinemo slave navigators. In these navigators, to avoid data loss, MOST or UDP stream data is read and buffered as fast as possible by dedicated prefetch threads.
RANGE	0 - INFINITE
DEFAULT VALUE	1024

CINEMO_OPTION_BUFFER_MAX_QUEUE_HISTORY_MS

TYPE	INTEGER
DESCRIPTION	In addition to future samples, the graph playback queue references also past samples needed for decoding up to the current playback position. With this option it is possible to apply a threshold for the maximum amount of past samples to buffer in the graph playback queue. Especially for livestreams with long video keyframe intervals and with already expected new keyframe it can be desireable to limit buffering of historic samples. This can avoid unnecessary latency when (re-)enabling video during an ongoing playback. If reference frames needed for decoding up to the current position are removed from the graph playback queue due to this option, playback of the respective stream can only resume after a new keyframe is received.
RANGE	0 - INFINITE
DEFAULT VALUE	0 (disabled)

CINEMO_OPTION_BUFFER_LIVESTREAM_PREBUFFER_MS

TYPE	INTEGER
DESCRIPTION	This option applies to livestream sources, e.g. streams played through the DMB navigator and streams provided through a CinemoLiveStream object. The option specifies how much data must be pre-buffered before the playback graph can start playback. The option also applies to a playback re-start after a buffer underrun.
RANGE	10 - 10000
DEFAULT VALUE	1500

CINEMO_OPTION_BUFFER_LOW_LATENCY_PREBUFFER_MS

TYPE	INTEGER
DESCRIPTION	This option applies to livestream sources, e.g. iAP or A2DP streams with low latency user interaction requirement. The option specifies how much data must be pre-buffered before the playback graph can start playback. The option also applies to a playback re-start after a buffer underrun.
RANGE	10 - 10000
DEFAULT VALUE	200

CINEMO_OPTION_BUFFER_CAPTURE_DELAY_MS

TYPE	INTEGER
DESCRIPTION	This option applies to capture sources, e.g. ALSA and V4L2. The option specifies how long video frames and audio samples are delayed so that no frame/sample arrives too late at their renderer. If the value is too low audio video sync cannot be maintained.
RANGE	0 - 500
DEFAULT VALUE	50

CINEMO_OPTION_BUFFER_FILE_CACHE_KB

TYPE	INTEGER
DESCRIPTION	Specify the internal cache size, in KB, for reading files.
RANGE	0 - 524288
DEFAULT VALUE	1024

CINEMO_OPTION_BUFFER_FILE_CACHE_PAGE_KB

TYPE	INTEGER
DESCRIPTION	Specify the internal cache page size, in KB, for reading files. This value is used as the size of "read" calls to the operating system.
RANGE	0 - 32768
DEFAULT VALUE	0

CINEMO_OPTION_BUFFER_FILE_WRITE_CACHE_KB

TYPE	INTEGER
DESCRIPTION	Specify the internal cache size, in KB, for writing files.
RANGE	4 - 64
DEFAULT VALUE	16

CINEMO_OPTION_BUFFER_FILE_WRITE_CACHE_SMALL_KB

TYPE	INTEGER
DESCRIPTION	Specify the internal cache size, in KB, for writing SPEEX files.
RANGE	1 - 64
DEFAULT VALUE	16

CINEMO_OPTION_SPEED_TRICK_MILLISECS

TYPE	INTEGER
DESCRIPTION	Specify the audio block size, in milliseconds, used for fast forward or backward play. This option applies only when the playback speed is negative or greater than the speed specified by the <i>CINEMO_OPTION_SPEED_MUTE</i> option.
NOTES	During trick play, the correct play speed is achieved by decoding short blocks of the audio stream at normal speed (1x), then skipping to the next block, which will be some distance proportional to the play speed and the block size specified here.
RANGE	50 - 60000
DEFAULT VALUE	1000

CINEMO_OPTION_SPEED_TIME_EVENTS_MILLISECS

TYPE	INTEGER
DESCRIPTION	Specify the time, in milliseconds, between periodic <i>CINEMO_EC_TIME</i> event notifications. These event notifications may be used, for example, to update the playback position slider. Note that the event is also posted aperiodically, such as when playback is paused or resumed.
RANGE	0 - 60000. A zero value will disable periodic <i>CINEMO_EC_TIME</i> notifications.
DEFAULT VALUE	250

CINEMO_OPTION_SPEED_STATUS_EVENTS_MILLISECS

TYPE	INTEGER
DESCRIPTION	Specify the time, in milliseconds, between periodic <i>CINEMO_EC_GRAPH_STATUS</i> events. Aperiodic events may also be posted for certain graph status changes.
RANGE	0 - 60000
DEFAULT VALUE	1000

CINEMO_OPTION_SPEED_MUTE

TYPE	INTEGER
DESCRIPTION	Specify the play speed, in units of 1000, after which audio time-stretch will be disabled. If the current playback speed is less than zero or greater than this value then either audio will be muted (if the track contains a video stream), or trick play mode will be entered (see <i>CINEMO_OPTION_SPEED_TRICK_MILLISECS</i> option).
RANGE	1000 - 8000
DEFAULT VALUE	3000

CINEMO_OPTION_SPEED_STILLIMAGE

TYPE	INTEGER
DESCRIPTION	Specify the time, in milliseconds, to display each image in a slideshow. "0" implies infinite duration.
RANGE	0 - INFINITE
DEFAULT VALUE	5000

CINEMO_OPTION_SPEED_STOP_ON_LIVESTREAM_OVERRUN

TYPE	BOOLEAN
DESCRIPTION	Specify the behaviour when encountering the end of available data during fast-forward playback of a live-stream. If enabled, playback will terminate when reaching the end of available data during fast-forward. If disabled, playback will enter CUE state and resume playback as soon as data is available.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_GAPLESS_ENABLE_FORMATCHANGE

TYPE	BOOLEAN
DESCRIPTION	Set to enable gapless playback of all files, regardless of audio or video stream format.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_GAPLESS_SPINUP_MILLISECS

TYPE	INTEGER
DESCRIPTION	Specify the time in milliseconds before starting playback that a spin-up command is sent to a gapless queued track.
NOTES	It may be the case that a track on a CD-ROM device is queued for gapless playback using <i>CINEMO_EC_OPEN_GAPLESS</i> . If the period of time between opening the track and starting playback is too long, the CD-ROM device may spin down, in which case there will be a delay when starting playback (while the device spins up again). This can be avoided by sending the spin-up command some time before starting playback.
RANGE	0 - 60000
DEFAULT VALUE	8000

CINEMO_OPTION_GAPLESS_OVERLAP_MS

TYPE	INTEGER
DESCRIPTION	Specify the time offset from the end of the current track, when gapless playback of the next track starts.
NOTES	For a seamless transition between gapless tracks, the playback of the next track needs to be started while the end of the current track is yet playing. This option controls the amount of time from the end of the current track, when the playback of the next track is started. When playback of the next track is started, the player will be associated with the next track, even though the played content yet relates to the current track. The interval should be chosen long enough so that there is enough time to supply data from the new track to the ongoing playback so to not cause any playback disruption.
RANGE	100 - 1000
DEFAULT VALUE	1000

CINEMO_OPTION_PLAYLIST_SKIP_ON_OPEN_ERROR

TYPE	BOOLEAN
DESCRIPTION	Enable automatic skipping to the next/ previous track in a playlist if opening of a track failed.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_PLAYLIST_SKIP_ON_PLAYBACK_ERROR

TYPE	BOOLEAN
DESCRIPTION	Enable automatic skipping to the next/ previous track in a playlist if playback of a track failed.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_SCAN_BYPASS_REPEAT_SINGLE

TYPE	BOOLEAN
DESCRIPTION	Enable bypassing repeat single mode when fast forward reaches end of file or backwards playback reaches beginning of file.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_SKIP_BYPASS_REPEAT_SINGLE

TYPE	BOOLEAN
DESCRIPTION	Enable bypassing repeat single mode when skipping to next/ previous track.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_SKIP_CANCEL_REPEAT_SINGLE

TYPE	BOOLEAN
DESCRIPTION	Turn off repeat single mode when skipping to next/ previous track.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_SKIP_UNPAUSE

TYPE	BOOLEAN
DESCRIPTION	Unpause playback when skipping to next/ previous track.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_PREVIOUS_TRACK_SEQUENTIAL

TYPE	BOOLEAN
DESCRIPTION	Always use sequential order when skipping to previous track.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_PREVIOUS_TRACK_FROMBEGINNING_MS

TYPE	INTEGER
DESCRIPTION	If playback position is already passed given time then skipping to previous track will seek to beginning of current file instead.
RANGE	0 - 60000. A value of zero will always do skipping to previous track.
DEFAULT VALUE	0

CINEMO_OPTION_PLAYLIST_SCAN_BW_RESUME_ON_BOT

TYPE	BOOLEAN
DESCRIPTION	Resume playing the current track at normal speed when backwards playback reaches beginning of file.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_SCAN_FW_RESUME_ON_EOT

TYPE	BOOLEAN
DESCRIPTION	Resume playing the next track at normal speed when fast forward reaches end of file.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_SCAN_BW_PAUSE_ON_BOP

TYPE	BOOLEAN
DESCRIPTION	Pause playing when backwards playback reaches beginning of playlist.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_SCAN_FW_PAUSE_ON_EOP

TYPE	BOOLEAN
DESCRIPTION	Pause playing when fast forward reaches end of playlist.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_SHUFFLE_MOVE_CURRENT_FIRST

TYPE	BOOLEAN
DESCRIPTION	Move currently playing track to first place while shuffling with ICinemoPlaylist::SetOrder() . Does not affect iAP and BT remote devices.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PLAYLIST_MAX_TRACKS

TYPE	INTEGER
DESCRIPTION	The upper limit of tracks allowed in a playlist. A value of 0 disables the limit.
RANGE	0 - INFINITE
DEFAULT VALUE	1000000

CINEMO_OPTION_PLAYLIST_ICU_LOCALE

TYPE	STRING
DESCRIPTION	Locale that should be used for sorting and index generation (utilizing the ICU library) of some specific VFS. If not specified, the UCA is used instead
RANGE	UTF-8
DEFAULT VALUE	en

CINEMO_OPTION_METADATA_MAX_SIZE

TYPE	INTEGER
DESCRIPTION	Specify the maximum size in bytes for a single metadata item (this includes both text and images). Metadata items larger than this will be ignored.
RANGE	0 - INFINITE
DEFAULT VALUE	1048576

CINEMO_OPTION_METADATA_MAX_IMAGE_RESOLUTION

TYPE	STRING
DESCRIPTION	Specify the maximum allowed resolution for image metadata. Image metadata with resolutions larger than this will be ignored. If only one integer is given, this number will be compared to the pixel count (i.e. the product of width and height). If two delimited integers are given, width and height will be compared separately.
RANGE	UTF-8. String with either one integer or two delimited integers. Allowed delimiters: ", . ; xx" Example: "1280x720"
DEFAULT VALUE	NULL

CINEMO_OPTION_METADATA_IMAGE_LIMIT

TYPE	INTEGER
DESCRIPTION	Limit the number of image metadata to be extracted. When processing ID3v2 style metadata, the picture type priority list (see CINEMO_OPTION_METADATA_IMAGE_PRC) will be taken into account when deciding which image metadata will be extracted.
RANGE	0 - INFINITE. A value of zero will disable the limit.
DEFAULT VALUE	0

CINEMO_OPTION_METADATA_IMAGE_PRIO

TYPE	STRING
DESCRIPTION	Prioritize picture types during image metadata processing.
RANGE	"p1, ..., pn" where each entry is either the picture type according to ID3v2 picture types or "*" for any type.
EXAMPLE	<p>"*" – No prioritization, use order of occurrence</p> <p>"3" – Only consider front cover</p> <p>"3,*" – Prioritize front cover, otherwise use order of occurrence</p>
DEFAULT VALUE	*

CINEMO_OPTION_METADATA_ENABLE_IMAGES

TYPE	BOOLEAN
DESCRIPTION	Enable images or cover art metadata to be extracted.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_METADATA_ENABLE_SNAPSHOTS

TYPE	BOOLEAN
DESCRIPTION	Enable snapshot metadata.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_METADATA_ENABLE_THUMBS

TYPE	BOOLEAN
DESCRIPTION	Enable thumbnail generation.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_METADATA_ENABLE_ID3_MERGING

TYPE	BOOLEAN
DESCRIPTION	Enable merging of ID3 metadata attributes, if both ID3v2 and ID3v1 are present in a single media file. If enabled, valid ID3v1 attributes not found in an also present ID3v2 tag will still be added to the metapool. If disabled, only the most recent version of ID3 tags found will be considered.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_METADATA_THUMB_TIMEOUT_MS

TYPE	INTEGER
DESCRIPTION	<p>Specify the maximum acceptable amount of time, in milliseconds, to wait for availability of a thumbnail. If the thumbnail generation process takes longer than the specified amount of time, or if there is an error during thumbnail generation, retrieval of a thumbnail is aborted. If thumbnail retrieval is aborted, the reason for it is stored as CinemoError code in the metapool item CINEMO_METANAME_THUMB_ERROR. In this case, a thumbnail may still be provided, but it is not guaranteed that the provided thumbnail is the same thumbnail as if there would have been no error.</p> <p>The special value 0 indicates that the thumbnail generation process should not terminate prematurely.</p>
RANGE	0 - INFINITE
DEFAULT VALUE	4000

CINEMO_OPTION_METADATA_DEFAULT_CHARSET

TYPE	STRING
DESCRIPTION	<p>Specify a comma separated list of character encodings for metadata types which do not explicitly specify character encoding. This applies for RIFF metadata (.wav, .avi), ID3 metadata in case of ID3v1 and non-numerical string metadata in ID3v2, if the default character set encoding type (0) is signaled. For non-numerical metadata first an ISO-8859-1 plausibility check is performed and if this succeeds, ISO-8859-1 will be used. Otherwise, the encoding from the list which yields the least amount of unrecognizable characters is finally chosen.</p>
RANGE	utf-8, utf-16-le, utf-16-be, utf-32-le, utf-32-be, iso-646, iso-8859-1, iso-8859-2, iso-8859-3, iso-8859-4, iso-8859-5, iso-8859-6, iso-8859-7, iso-8859-8, iso-8859-9, iso-8859-10, iso-8859-11, iso-8859-13, iso-8859-14, iso-8859-15, iso-8859-16, cp-037, cp-437, cp-500, cp-720, cp-737, cp-775, cp-850, cp-852, cp-855, cp-857, cp-858, cp-860, cp-861, cp-862, cp-863, cp-864, cp-865, cp-866, cp-869, cp-874, cp-875, cp-1026, cp-1250, cp-1251, cp-1252, cp-1253, cp-1254, cp-1255, cp-1256, cp-1257, cp-1258, cp-932, cp-936, cp-949, cp-950, euc-jp, euc-kr, koi8-r, koi8-u, gb-18030, tis-620, iso-2022-cn, iso-2022-kr, hkscs, gb-2312
DEFAULT VALUE	NONE

CINEMO_OPTION_METADATA_PLAYLIST_CACHE_KB

TYPE	INTEGER
DESCRIPTION	Specify the playlist client-side metadata cache size, in KB.
RANGE	0 - 32768. A zero value indicates unlimited size.
DEFAULT VALUE	10240

CINEMO_OPTION_UPNP_EVENT_CALLBACK_PORT

TYPE	STRING
DESCRIPTION	A range of TCP ports where the UPnP client will open a HTTP server to listen for events from the UPnP server.
EXAMPLE	<p>"0" – Random port</p> <p>"40000" – Manually assigned single port</p> <p>"40000-40010" – Manually assigned port range</p>
RANGE	"x" or "x-y" where x, y are within 0–65535. A value of zero or NULL will use random ports.
DEFAULT VALUE	NULL

CINEMO_OPTION_NETWORKMONITOR_AUTODETECTION

TYPE	BOOLEAN
DESCRIPTION	Enable automatic detection of network changes
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_IP_DSCP

TYPE	INTEGER
DESCRIPTION	Specify the DSCP field value (6bit DSCP value, not including ECN) used for all TCP/IP and UDP traffic.
RANGE	0 - 63
DEFAULT VALUE	0

CINEMO_OPTION_GLOBAL_POSITIONING_SUPPRESS_COORDINATES_DISPLAY

TYPE	BOOLEAN
DESCRIPTION	This option allows to turn off displaying of coordinates from global positioning systems. If set to true Cinemo will either output some place holder value or suppress further output which might contain the coordinates values. This applies also to logging output.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

11.2 HTTP Options

CINEMO_OPTION_HTTP_TIMEOUT

TYPE	INTEGER
DESCRIPTION	Specify a timeout value, in milliseconds, after which file open operations will fail. Only applies to HTTP and FTP files.
RANGE	0 - INFINITE. A value of zero will disable the timeout.
DEFAULT VALUE	30000

CINEMO_OPTION_HTTP_TIMEOUT_LOW_LEVEL

TYPE	INTEGER
DESCRIPTION	Specify a timeout value, in milliseconds, after which an HTTP connection attempt will be considered as failed. Retry logic would then be active if enabled.
RANGE	0 - INFINITE. A value of zero will disable the timeout.
DEFAULT VALUE	10000

CINEMO_OPTION_HTTP_RETRIES_LOW_LEVEL

TYPE	INTEGER
DESCRIPTION	Specify the number of retries. HTTP operations will be retried after the time set by option <i>CINEMO_OPTION_HTTP_TIMEOUT_LOW_LEVEL</i> has expired.
RANGE	0 - INFINITE. A value of zero will disable retries.
DEFAULT VALUE	0

CINEMO_OPTION_HTTP_TIME_WAIT_RETRY

TYPE	INTEGER
DESCRIPTION	Specify a value, in milliseconds, of time to wait before retrying HTTP operations. Note, this option is only enabled if <i>CINEMO_OPTION_HTTP_RETRIES_LOW_LEVEL</i> is greater than zero.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_HTTP_TIME_WAIT_DATA

TYPE	INTEGER
DESCRIPTION	Specify a timeout value, in milliseconds, after which a lack of data from an HTTP server will be considered a broken connection.
RANGE	0 - INFINITE
DEFAULT VALUE	10000

CINEMO_OPTION_HTTP_MAX_SERVER_CLIENTS

TYPE	INTEGER
DESCRIPTION	Specify the maximum number of concurrent client connections served at a time by HTTP server(s) used in Cinemo.
RANGE	1 - 10000
DEFAULT VALUE	100

CINEMO_OPTION_HTTP_MAX_REDIRECTIONS

TYPE	INTEGER
DESCRIPTION	Specify the maximum number of redirections allowed in case of HTTP playback (http://protocol). If the maximum number of redirections has been exceeded, <i>CinemoErrorRedirection</i> will be returned and HTTP playback will be stopped.
DEFAULT VALUE	-1

CINEMO_OPTION_HTTP_REDIRECT_CALLBACK

TYPE	CALLBACK
DESCRIPTION	Specify a callback function to be called for HTTP redirections. If the callback function returns a value not equal to 0, the redirection is not performed and instead a <i>CinemoErrorRedirection</i> is returned by the function used to open the original URL. The callback function will receive a reference to a <i>CinemoRedirectParams</i> struct including: szurl_target: target URL of the redirection szurl_original: original URL to the site initiating this redirection redirect_count: number of redirections
RANGE	Callback function pointer of type <i>CinemoRedirectCallback</i>
DEFAULT VALUE	NULL
NOTES	If multiple redirections are chained (the target URL is another redirection), the callback will still only be called once. In this case <i>szurl_target</i> will contain the final url of the chained redirections. Please note that the callback will not be called in case the number of chained redirections exceeds the limit set via the option <i>CINEMO_OPTION_HTTP_MAX_REDIRECTIONS</i> .

CINEMO_OPTION_HTTP_SOURCEPORT

TYPE	STRING
DESCRIPTION	Specify the source port range which will be used for HTTP connections. If set to NULL, the usage of source ports will not be restricted and random ports (depending on OS) will be used.
RANGE	"x-y" where x, y are in valid http port range.
DEFAULT VALUE	NULL

CINEMO_OPTION_HTTP_PROXY_URL

TYPE	STRING
DESCRIPTION	Specify the proxy server and port number for all HTTP and FTP connections. Several proxy protocols are supported and can be selected by URL scheme. The full syntax supported is: [protocol://][user:password@]proxyhost[:port]
Supported Protocols:	
	“http” - HTTP proxy (this is the default)
	“socks4” - SOCKS4 proxy
	“socks4a” - SOCKS4 proxy with host name resolution done by proxy
	“socks5” - SOCKS5 proxy
	“socks5h” - SOCKS5 proxy with host name resolution done by proxy
Examples:	
	“192.168.1.200:8080”
	“socks5://localhost”
RANGE	UTF-8 encoded URL
DEFAULT VALUE	NULL

CINEMO_OPTION_HTTP_SSL_CA

TYPE	STRING
DESCRIPTION	Specifies a CA file located at the given path.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_HTTP_SSL_CERT

TYPE	STRING
DESCRIPTION	Specifies a client certificate file located at the given path.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_HTTP_SSL_KEY

TYPE	STRING
DESCRIPTION	Specifies a client private key file located at the given path.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_HTTP_SSL_PASSWORD

TYPE	STRING
DESCRIPTION	Specifies a passphrase for the private key.
RANGE	UTF-8
DEFAULT VALUE	NULL

11.3 SSDP Options

CINEMO_OPTION_SSDP_CUSTOMER_PRODUCT_ID	
TYPE	STRING
DESCRIPTION	<p>Specify the product-unique string to append to the Cinemo product ID in the UPnP server field. This allows for unique identification during the UPnP discovery process among Cinemo instances, such as head units and rear seat units. It can also be useful to identify the brand of the vehicle for customizations. If not set or set to NULL, no product-unique string will be appended.</p> <p>On the client side the UPnP server field can be obtained through the VFS metapool by looking up <i>CINEMO_METANAME_VFS_SSDP_SERVER</i>.</p>
EXAMPLE	<p>"RACE-HU" – SSDP server field could be "androidndk_Generic/1.0 UPnP/1.0 Cinemo-RACE-HU/1.33.0.23041" (Race is a fake vehicle brand and HU stands for Head Unit; note that the "-" dash is automatically inserted before the string)</p> <p>"" (NULL) – SSDP server field example above would be "androidndk_Generic/1.0 UPnP/1.0 Cinemo/1.33.0.23041"</p>
NOTES	The customer product ID of a Cinemo mobile app begins with MOBILE. By checking if the Cinemo product ID starts with "Cinemo-MOBILE" a Cinemo app can be identified. The following server field identifies a mobile app: "androidndk_Generic/1.0 UPnP/1.0 Cinemo-MOBILE-TAB /1.33.0.23041"
RANGE	UTF8
CINEMO_OPTION_SSDP_MSEARCH_INTERVAL	
TYPE	INTEGER
DESCRIPTION	<p>Specify an interval between UPnP SSDP search requests. Search broadcasts will be repeated continuously.</p> <p>The unit is seconds.</p>
RANGE	0 - INFINITE. A value of zero will disable searches in regular intervals.
DEFAULT VALUE	30

CINEMO_OPTION_SSDP_MSEARCH_AUTO_EXPIRE_INTERVAL

TYPE	INTEGER
DESCRIPTION	Specify an interval after which UPnP services are considered out-dated and notified as unavailable. This value precedes any service properties provided by UPnP servers that indicate a longer expiration. It is recommended to set this interval to higher value than the SSDP_MSEARCH_INTERVAL.
	The unit is seconds.
RANGE	0 - INFINITE. A value of zero will disable early expiration of UPnP services.
DEFAULT VALUE	40

CINEMO_OPTION_SSDP_MSEARCH_REPEAT_COUNT

TYPE	INTEGER
DESCRIPTION	Specify how many UPnP SSDP search broadcasts will be sent at a time. Repetition is highly recommended due to the unreliable nature of UDP broadcasts.
RANGE	1 - 20.
DEFAULT VALUE	3

CINEMO_OPTION_SSDP_SEARCH_NIC

TYPE	STRING
DESCRIPTION	Name of network interface to use for SSDP search queries.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_SSDP_SEARCH_PORT

TYPE	STRING
DESCRIPTION	UDP port the SSDP client uses for incoming search responses.
EXAMPLE	<p>"0" – Random port</p> <p>"40000" – Manually assigned single port</p> <p>"40000-40010" – Manually assigned port range</p>
RANGE	"x" or "x-y" where x, y are within 0–65535. A value of zero or NULL will use random ports.
DEFAULT VALUE	NULL

11.4 Protection Options

CINEMO_OPTION_PROTECTION_KEY_STORAGE_CONTAINER_PATH

TYPE	STRING
DESCRIPTION	Absolute path to the key storage container (KSC) file created with the Cinemo_KSC command line tool.
RANGE	Host system path syntax
DEFAULT VALUE	<Cinemo install directory>/lib/key/ksc0.bin

CINEMO_OPTION_PROTECTION_KEY_STORAGE_CONTAINER_TAG

TYPE	STRING
DESCRIPTION	Expected tag of the KSC. KSCs will only be accepted if they have no tag or the expected TAG. This option is not saved to the configuration file. It needs to be set via a SetOption call to the Cinemo API.
RANGE	UTF8
DEFAULT VALUE	NULL

CINEMO_OPTION_PROTECTION_DTCP_KEY_TYPE

TYPE	ENUM
DESCRIPTION	Expected type of DTCP key. To be able to use facsimile device certificates instead of DTLA device certificates, this option allows to execute a declaration of intention to do so.
RANGE	“production”, “facsimile”
DEFAULT VALUE	“production”

CINEMO_OPTION_PROTECTION_DTCP_FUNCTION

TYPE	ENUM
DESCRIPTION	Specify the DTCP-IP function for distributed playback. Sources can distribute protected content, sinks can receive and render protected content.
RANGE	“none”, “source”, “sink”
DEFAULT VALUE	“none”

CINEMO_OPTION_PROTECTION_DTCP_SOURCE_AUTHSERVER_URL

TYPE	STRING
DESCRIPTION	Specify the TCP URL for the DTCP-IP authentication server on DTCP sources for distributed playback.
EXAMPLE	"tcp://192.168.0.1:6050" - Authentication server on port 6050 and IP 192.168.0.1. "tcp://@:6050" - Authentication server on port 6050 and all compatible NICs. "tcp://@" - Authentication server on any port and all compatible NICs.
RANGE	UTF-8
DEFAULT VALUE	"tcp://@"

CINEMO_OPTION_PROTECTION_WRITABLE_STORAGE_PATH

TYPE	STRING
DESCRIPTION	Path to a folder where the protection system can place its persistent data. Used for holding system renewability message information as required by the DTLA. The system renewability messages have an overall maximum space requirement of 1024 bytes.
RANGE	Host system path syntax
DEFAULT VALUE	NULL

CINEMO_OPTION_PROTECTION_ENABLE_ACCEPT_CMI

TYPE	BOOLEAN
DESCRIPTION	Specify whether changes in content management information (CMI) need to be accepted. If enabled, the rendering will be disabled until the CMI information passed by the event CINEMO_EC_CMI are accepted by calling ICinemoPlayer::AcceptCMI() with the appropriate CMI value.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_PROTECTION_DTCP_AUTH_RESULT_CALLBACK

TYPE	CALLBACK
DESCRIPTION	<p>Specify a callback function to be notified of DTCP authentication results. The callback function will be called when DTCP authentication succeeds, fails for some reason or is cancelled by the peer.</p> <p><i>CinemoNoError</i> will be stored in params.hr if and only authentication succeeded.</p> <p><i>CinemoErrorBadCertificate</i> will be stored in params.hr if and only if the peer's certificate could not be verified successfully.</p> <p><i>CinemoErrorKeyRevoked</i> will be stored in params.hr if and only if the peer's certificate has been revoked by the DTLA.</p> <p>Various other Cinemo error codes might be returned.</p> <p>params.peer stores the IP address and port of the DTCP sink or source the authentication was going on with.</p>
RANGE	Callback function pointer of type CinemoDTCPAuthenticationResultCallback
DEFAULT VALUE	NULL
NOTES	It is possible that the function will be called more than once per authentication process in case of failure.

11.5 Video Options

CINEMO_OPTION_VIDEO_ENABLE

TYPE	BOOLEAN
DESCRIPTION	Enable video decoding and rendering. If disabled, video decoder and renderer resources will NOT be allocated. Files containing video will still play, but only audio streams be decoded and rendered.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_VIDEO_HARDWARE_DECODE

TYPE	BOOLEAN
DESCRIPTION	Enable hardware accelerated video decoding. If enabled, hardware acceleration features of the target platform will be used if supported; otherwise Cinemo will fall back to software video decoding.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_VIDEO_HARDWARE_DEINTERLACE

TYPE	BOOLEAN
DESCRIPTION	Enable hardware accelerated video de-interlacing.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_VIDEO_HARDWARE_MIX

TYPE	BOOLEAN
DESCRIPTION	Enable hardware accelerated video mixing, e.g. of subpicture or other overlay streams.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_VIDEO_DEVICE_ID

TYPE	STRING
DESCRIPTION	Specify the name and parameters of the video device for rendering. This will vary depending on the target platform. Use "?hwnd=%p" to pass a native window handle to be used for rendering.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_VIDEO_CODEC_PARAMETERS

TYPE	STRING
DESCRIPTION	Specify URL parameter string for video codec plugin. This will vary depending on the target platform.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_VIDEO_ASPECT

TYPE	ENUM
DESCRIPTION	Specify the video aspect ratio mode, which controls the way video will appear in a target display rectangle.
RANGE	Preserve: preserves the original aspect ratio, black borders may be drawn on the left/right or top/bottom of the display rectangle. Fill: preserves the original aspect ratio, and zoom in as much that black borders are no longer needed. Video content will fill the whole display rectangle. Regions of video which fall outside of the display rectangle are cropped. Stretch: ignore the video aspect ratio, and scale to exactly fit the display rectangle. Native: ignore the video aspect ratio and render directly to the display with a 1:1 pixel relationship. Falling back to vertical and/or horizontal stretch mode when regions of video would fall outside of the display rectangle. Wall: special mode for multiple display video wall mode. Similar to preserve mode, but relative source rectangle specifying position inside the video wall.
DEFAULT VALUE	Preserve

CINEMO_OPTION_VIDEO_FRAME_DROP_THRESHOLD

TYPE	INTEGER
DESCRIPTION	Specify the threshold time, in milliseconds, after which late video frames will be dropped (or some other quality control method will be activated, whichever is appropriate for the currently decoding video stream).
RANGE	0 - 1000. A value of zero will disable frame dropping.
DEFAULT VALUE	50

CINEMO_OPTION_VIDEO_DELAY_MS

TYPE	INTEGER
DESCRIPTION	Specify the time, in milliseconds, to delay video in order to synchronize with audio for this distributed playback client.
RANGE	-100,000 to 100,000
DEFAULT VALUE	0

CINEMO_OPTION_VIDEO_EXTRA_SURFACES

TYPE	INTEGER
DESCRIPTION	Specify the number of additional video decoding surfaces which will be allocated, beyond those required for decoding. More surfaces allow video frames to be decoded further ahead of their rendering time, which may reduce jitter caused by some video frames taking longer to decode than others.
RANGE	0 - 32. A value of zero will disable the allocation of additional video surfaces.
DEFAULT VALUE	0

CINEMO_OPTION_VIDEO_CONTENT_STEREO_MODE

TYPE	ENUM
DESCRIPTION	Forces source video to be treated according to this setting.
RANGE	None,SideBySide,TopBottom
DEFAULT VALUE	None

CINEMO_OPTION_VIDEO_DISPLAY_STEREO_MODE

TYPE	ENUM
DESCRIPTION	Specifies the stereo mode of the monitor.
RANGE	None,SideBySide,TopBottom
DEFAULT VALUE	None

CINEMO_OPTION_VIDEO_DISPLAY_ASPECT_X**CINEMO_OPTION_VIDEO_DISPLAY_ASPECT_Y**

TYPE	INTEGER
DESCRIPTION	Specify the physical pixel aspect ratio of the display. For LCD displays, the pixel aspect ratio is typically 1:1. This is taken into account when calculating the video target rectangle (see the <i>CINEMO_OPTION_VIDEO_ASPECT</i> option).
RANGE	0 - 10000
DEFAULT VALUE	1

CINEMO_OPTION_VIDEO_MAX_H264_LEVEL

TYPE	STRING
DESCRIPTION	Specify the maximum supported level for decoding H.264 streams. For streams which exceed the specified level, playback will abort with the error <i>CinemoErrorMaxH264Level</i> .
RANGE	UTF-8. String with level limit. Example: "4.2"
DEFAULT VALUE	NULL

CINEMO_OPTION_VIDEO_MAX_RESOLUTION

TYPE	STRING
DESCRIPTION	Specify the maximum supported resolution for video streams. For streams which exceed the specified resolution, a playback attempt will abort with the error <i>CinemoErrorMaxResolution</i> . Depending on the license agreement with Cinemo it might not be possible to modify this option.
RANGE	UTF-8. String with two delimited integers. Allowed delimiters: “, . ; ×” Example: “1280x720”
DEFAULT VALUE	NULL

CINEMO_OPTION_VIDEO_NAVIGATOR_RESOLUTION_CHECK

TYPE	BOOLEAN
DESCRIPTION	Specify whether <i>CINEMO_OPTION_VIDEO_MAX_RESOLUTION</i> should be checked already when opening a video stream (e.g. for metadata retrieval). If TRUE, an attempt to open a video stream with too high resolution will fail with the error <i>CinemoErrorMaxResolution</i> .
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_VIDEO_NAVIGATOR_IMAGE_CONSTRAINT_CHECK

TYPE	BOOLEAN
DESCRIPTION	Specify whether <i>CINEMO_OPTION_VIDEO_MAX_IMAGE_RESOLUTION</i> and <i>CINEMO_OPTION_VIDEO_MAX_IMAGE_SIZE</i> should be checked already when opening an image file (e.g. for metadata retrieval). If TRUE, an attempt to open an image file with too high resolution or too big file size will fail with the error <i>CinemoErrorMaxResolution</i> .
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_VIDEO_MAX_IMAGE_RESOLUTION

TYPE	STRING
DESCRIPTION	Specify the maximum supported resolution for image files. For image files which exceed the specified resolution, a playback attempt will abort with the error <i>CinemoErrorMaxResolution</i> .
RANGE	UTF-8. String with two delimited integers. Allowed delimiters: “, . ; ×” Example: “1280x720”
DEFAULT VALUE	NULL (resolution check disabled)

CINEMO_OPTION_VIDEO_MAX_IMAGE_SIZE

TYPE	INTEGER
DESCRIPTION	Specify the maximum supported size in bytes for image files. For image files which exceed the specified size, a playback attempt will abort with the error <i>CinemoErrorMaxResolution</i> .
RANGE	0 - INFINITE
DEFAULT VALUE	0 (size check disabled)

CINEMO_OPTION_VIDEO_MAX_RENDERED_FPS

TYPE	INTEGER
DESCRIPTION	Specify the maximum rendered frames per second (1000=1fps). In case of high frame rate content, when the renderer is limited to LCD vertical synchronization, this option helps avoid dropping blocks of frames in the decoder. Instead, single frames are dropped at the renderer without interfering with later frame decoding.
RANGE	0 - 1024000. A value of zero disables this option.
DEFAULT VALUE	0

CINEMO_OPTION_VIDEO_IMAGE_BACKGROUND_COLOR

TYPE	STRING
DESCRIPTION	Specify the background color for images with alpha channel or transparency information. This option works in combination with <i>CINEMO_OPTION_VIDEO_FORCE_IMAGE_BACKGROUND_COLOR</i> . See the explanations there for more details.
RANGE	HTML color code. E.g. 'ff0000' for a red background. An empty value will trigger use of a internally pre-defined background color.
DEFAULT VALUE	NULL

CINEMO_OPTION_VIDEO_FORCE_IMAGE_BACKGROUND_COLOR

TYPE	BOOLEAN
DESCRIPTION	This option works in combination with the option <i>CINEMO_OPTION_VIDEO_IMAGE_BACKGROUND_COLOR</i> . The background color specified there will normally only be used if the image format doesn't define a default background color to be used. If TRUE, default background color information in the image will be ignored.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_VIDEO_NATIVE_ASPECTMODE_SCALING

TYPE	BOOLEAN
DESCRIPTION	Enable video scaling in CINEMO_ASPECTMODE_NATIVE aspect mode to match display aspect ratio.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_VIDEO_DIVXCOMPATIBILITY_ENABLE_AVI

TYPE	BOOLEAN
DESCRIPTION	Enable support of AVI files with DivX-associated FourCC identification code.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_VIDEO_DIVXCOMPATIBILITY_ENABLE_MKV

TYPE	BOOLEAN
DESCRIPTION	Enable support of Matroska files which have been created by the DivX muxing application.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_VIDEO_DIVXCOMPATIBILITY_ENABLE_XSUB

TYPE	BOOLEAN
DESCRIPTION	Enable support for DivX subtitle streams (embedded in AVI).
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_VIDEO_DIVXCOMPATIBILITY_SKIP_STREAMMARKER_CHECK

TYPE	BOOLEAN
DESCRIPTION	Enables skipping of the check for DivX embedded markers in the video bitstream. If skipping of the marker check is enabled, playback will not stop if DivX markers are found in the video bitstream.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_VIDEO_CAPTURE_DEVICE_ID

TYPE	STRING
DESCRIPTION	Specify the name of the video capture device. This device will be used for capturing from video inputs.
RANGE	UTF-8
DEFAULT VALUE	"" (empty)

11.6 Audio Options

CINEMO_OPTION_AUDIO_ENABLE

TYPE	BOOLEAN
DESCRIPTION	Enable audio decoding and rendering. If disabled, audio decoder and renderer resources will NOT be allocated. Files containing audio will still play, but only video streams be decoded and rendered.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_AUDIO_ENABLE LOSSLESS_CODECS

TYPE	INTEGER
DESCRIPTION	Specify if playback of lossless audio codecs is enabled or not. The only audio codec which is affected by this option is DTS-HD MA. This setting can be used to cap memory consumption.
RANGE	0: playback of any audio or video file containing DTS-HD MA audio streams will fail with <i>CinemoErrorLosslessAudioCodec</i> error value. 1: only the lossy part of a DTS-HD MA audio stream will be played. 2: DTS-HD MA is allowed to be played, including the lossless part.
DEFAULT VALUE	2

CINEMO_OPTION_AUDIO_JITTER_THRESHOLD

TYPE	INTEGER
DESCRIPTION	Specify the threshold, in milliseconds, after which jitter in audio timestamps will be compensated. Audio timestamp jitter may occur if the source audio timestamps are not or cannot be generated accurately (as for example with some AVI-files). In this case the jitter will be compensated for in the resulting decompressed audio stream by adding silence in case of gaps or by removing samples in case of overlap. The jitter threshold should be set to a value greater than the expected timestamp jitter, but less than the point when audio/video drift becomes perceptible.
RANGE	0 - 1000. A value of zero will disable audio jitter compensation.
DEFAULT VALUE	50

CINEMO_OPTION_AUDIO_DELAY_MS

TYPE	INTEGER
DESCRIPTION	Specify the time, in milliseconds, to delay audio for this client in order to synchronize with video.
RANGE	-100,000 to 100,000
DEFAULT VALUE	0

CINEMO_OPTION_AUDIO_SPDIF_MODE

TYPE	ENUM
DESCRIPTION	Configure S/PDIF compatible output of AC-3 and DTS compressed streams.
RANGE	<p>"Disabled" - S/PDIF output disabled (only regular PCM output)</p> <p>"Enabled" - If possible, output AC-3 and DTS compressed streams to the audio device</p> <p>"Secondary" - Audio is output as PCM, but an additional audio renderer is added which will be used to output AC-3 and DTS compressed streams, if possible.</p>
Note: S/PDIF playback is not possible with live data sources	
DEFAULT VALUE	"Disabled"

CINEMO_OPTION_AUDIO_RENDERER_BUFFER_MS

TYPE	INTEGER
DESCRIPTION	Manually configure the internal audio renderer buffer size.
RANGE	0 - 5000. A value of zero will let Cinemo automatically control the internal audio renderer buffer size, optimizing it to the media source and playback options value. This value should only be changed after consultation with Cinemo.
DEFAULT VALUE	0

CINEMO_OPTION_AUDIO_HTTP_PREFETCH_BUFFER_MS

TYPE	INTEGER
DESCRIPTION	Audio renderer buffer size for HTTP based audio.
RANGE	100-10000. The value of this option will be used if and only if <i>CINEMO_OPTION_AUDIO_RENDERER_BUFFER_MS</i> is zero.
DEFAULT VALUE	1000

CINEMO_OPTION_AUDIO_DEVICE_ID

TYPE	STRING
DESCRIPTION	Specify the name of the audio device for rendering. This will vary depending on the target platform. On Linux, the default audio device is "default".
RANGE	UTF-8
DEFAULT VALUE	"default"

CINEMO_OPTION_AUDIO_SECONDARY_DEVICE_ID

TYPE	STRING
DESCRIPTION	Specify the name of the secondary audio device. This device will be used for S/PDIF compatible output if <i>CINEMO_OPTION_AUDIO_SPDIF_MODE</i> is set to "Secondary"
RANGE	UTF-8
DEFAULT VALUE	"" (empty)

CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_ID

TYPE	STRING
DESCRIPTION	Specify the name of the audio capture device. This device will be used for audio capturing for Android Auto and CarPlay sessions.
RANGE	UTF-8
DEFAULT VALUE	"" (empty)

CINEMO_OPTION_AUDIO_DEVICE_SAMPLERATE

TYPE	INTEGER
DESCRIPTION	Specify the audio device sampling rate, in units of samples per second. All rendered audio will be re-sampled to the value specified here, before sending to the audio device.
RANGE	0 - 192000. A value of zero will disable samplerate conversion. In this case the audio device must be capable of DYNAMIC sample rate changes - e.g. the sample rate delivered to the audio device will be as encoded in the currently playing audio stream, and may change from one stream to another.
DEFAULT VALUE	0

CINEMO_OPTION_AUDIO_DEVICE_CHANNELS

TYPE	ENUM
DESCRIPTION	Specify the audio device channel configuration. All rendered audio will be re-mixed either up or down to the channel configuration specified here.
RANGE	<p>“Unspecified”</p> <p>“Mono”</p> <p>“Stereo”</p> <p>“L-C-R”</p> <p>“L-C-R-CS”</p> <p>“L-C-R-LS-RS”</p> <p>“5.1”</p> <p>“6.1”</p> <p>“7.1”</p> <p>A value of “Unspecified” will disable channel re-mixing. In this case the audio device must be capable of DYNAMIC channel changes - e.g. the channels delivered to the audio device will be as encoded in the currently playing audio stream, and may change from one stream to another.</p>
DEFAULT VALUE	“Unspecified”

CINEMO_OPTION_AUDIO_DEVICE_MAX_CHANNELS

TYPE	INTEGER
DESCRIPTION	Specify the maximum allowed audio output channel count. Audio streams with more channels will be downmixed. The upper limit for the allowed range is dependent on the license agreement with Cinemo.
RANGE	1 - maximum channel count according to license agreement with Cinemo
DEFAULT VALUE	maximum channel count according to license agreement with Cinemo

CINEMO_OPTION_AUDIO_DEVICE_SAMPLETYPE

TYPE	ENUM
DESCRIPTION	Specify the audio device sample type. All rendered audio samples will be converted to the format specified here.
RANGE	<p>"Unspecified"</p> <p>"U8" - Unsigned 8 bit</p> <p>"S8" - 8 bit</p> <p>"S16LE" - 16 bit Little Endian</p> <p>"S16BE" - 16 bit Big Endian</p> <p>"S24LE" - 24 bit Little Endian</p> <p>"S24BE" - 24 bit Big Endian</p> <p>"S32LE" - 32 bit Little Endian</p> <p>"S32BE" - 32 bit Big Endian</p> <p>"F32LE" - Float 32 bit Little Endian</p> <p>"F32BE" - Float 32 bit Big Endian</p> <p>"F64LE" - Float 64 bit Little Endian</p> <p>"F64BE" - Float 64 bit Big Endian</p>
	A value of "Unspecified" will attempt to use the sample type that is delivered by the audio decoder, if it is supported by the specific audio device. Else, the audio device will determine a reasonable sample type to be used by itself.
DEFAULT VALUE	"Unspecified"

CINEMO_OPTION_AUDIO_DEVICE_PERIOD_MS

TYPE	INTEGER
DESCRIPTION	Specify the period, in milliseconds, for this audio device. This is the smallest size of buffer that the audio device can handle. This value should be a divisor of <i>CINEMO_OPTION_AUDIO_DEVICE_BUFFER_MS</i> , such that a buffer consists of some integer number of periods. This value should only be changed after consultation with Cinemo.
RANGE	10 - 1000
DEFAULT VALUE	10

CINEMO_OPTION_AUDIO_DEVICE_BUFFER_MS

TYPE	INTEGER
DESCRIPTION	Specify the buffer size, in milliseconds, for this audio device. This value is used to calculate the sample buffer size for sending PCM data to the audio device. This value should be a multiple of <i>CINEMO_OPTION_AUDIO_DEVICE_PERIOD_MS</i> , such that a buffer consists of some integer number of periods. This value should only be changed after consultation with Cinemo.
RANGE	10 - 1000
DEFAULT VALUE	100

CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_PERIOD_MS

TYPE	INTEGER
DESCRIPTION	Specify the period, in milliseconds, for the audio capture device. This is the smallest size of buffer that the audio capture device can handle. This value should be a divisor of <i>CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_BUFFER_MS</i> , such that a buffer consists of some integer number of periods. This value should only be changed after consultation with Cinemo.
RANGE	10 - 1000
DEFAULT VALUE	25

CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_BUFFER_MS

TYPE	INTEGER
DESCRIPTION	Specify the buffer size, in milliseconds, for the audio capture device. This value is used to calculate the sample buffer size for receiving PCM data from the audio capture device. This value should be a multiple of <i>CINEMO_OPTION_AUDIO_CAPTURE_DEVICE_PERIOD_MS</i> , such that a buffer consists of some integer number of periods. This value should only be changed after consultation with Cinemo.
RANGE	10 - 1000
DEFAULT VALUE	50

CINEMO_OPTION_AUDIO_DEVICE_DRIFT_THRESHOLD

TYPE	INTEGER
DESCRIPTION	Specify the threshold, in milliseconds, after which audio drift will be compensated. Audio drift may occur if an audio rendering device runs at a different clock rate than the stream clock. In this case, the drift is compensated for by adjusting the audio datarate, thus advancing or delaying the audio to re-gain synchronization. The drift threshold should be set to a value greater than the expected clock jitter, but less than the point when audio/video drift becomes perceptible.
RANGE	0 - 1000. A value of zero will disable audio drift compensation.
DEFAULT VALUE	10

CINEMO_OPTION_AUDIO_DEVICE_ENABLE_HW_PAUSE

TYPE	BOOLEAN
DESCRIPTION	Set this option to enable pausing in hardware, if supported. If enabled with TRUE, the audio device will be commanded to pause and keep its buffer state. If disabled with FALSE, the audio playback will stop and buffers will be flushed, and on playback resume, the samples will be resent to the audio device.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_AUDIO_DEVICE_ENABLE_HW_VOLUME

TYPE	BOOLEAN
DESCRIPTION	Set this option to enable volume change in hardware, if supported. If enabled with TRUE, the audio device will be commanded to change the volume and keep its buffer state. If disabled with FALSE, the audio volume will be controlled by software-scaling of the PCM samples that are delivered to the audio device.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_AUDIO_DEVICE_ENABLE_HW_MIXING

TYPE	BOOLEAN
DESCRIPTION	Set this option to enable device level audio mixing. If enabled with TRUE, mixing of individual audio streams will be left to the audio device. The necessity of mixing exists for example in the case of BD interactive audio effect rendering or if there is audio playback with multiple Cinemo players from within one process. If disabled with FALSE, mixing of individual audio streams will be performed on Cinemo side and there will be only one resulting mixed stream per process that is provided to the audio device. It is recommended to leave this option to TRUE, unless mixing functionality cannot be provided by the audio device.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_AUDIO_STEREO_DOWNMIX

TYPE	ENUM
DESCRIPTION	Set this value to specify the type of stereo downmixing applied for Dolby and DTS streams. This requires a stereo channel configuration and a source with more than two channels.
RANGE	<p>Unspecified – stereo downmixing is determined by the bitstream.</p> <p>Stereo – standard stereo downmixing</p> <p>SurroundCompatible – creation of a Dolby Surround compatible stereo signal, i.e. a signal for which rear channel information can be recreated by Dolby Surround compatible receivers.</p>
DEFAULT VALUE	Stereo

CINEMO_OPTION_AUDIO_LFE_DOWNMIX

TYPE	ENUM
DESCRIPTION	Set this value to specify the type of LFE downmixing applied. This option requires CINEMO_OPTION_AUDIO_STEREO_DOWNMIX to be enabled to take effect.
RANGE	<p>Unspecified – LFE channel inclusion is determined by the bitstream.</p> <p>On – LFE channel is included in the stereo channels.</p> <p>Off – LFE channel is discarded when creating the stereo downmix.</p>

CINEMO_OPTION_AUDIO_RESAMPLE_QUALITY

TYPE	ENUM
DESCRIPTION	<p>Specify the audio resampling quality.</p> <p>This option applies only when it is required to resample audio before sending to the audio device (see <i>CINEMO_OPTION_AUDIO_DEVICE_SAMPLERATE</i>).</p>
RANGE	<p>Linear – linear interpolation algorithm is applied, which is very CPU efficient but may be subject to aliasing. Not recommended to use!</p> <p>Low – band-limited interpolation algorithm with low precision is applied.</p> <p>Normal – band-limited interpolation algorithm with normal precision is applied.</p> <p>High – band-limited interpolation algorithm with high precision is applied.</p>
DEFAULT VALUE	Normal

CINEMO_OPTION_AUDIO_RATECHANGE_QUALITY

TYPE	ENUM
DESCRIPTION	<p>Specify the audio resampling quality for asynchronous ratechange. This option controls the accuracy of the audio resampling that is applied in order to compensate for drift of the current audio playback position from the playback clock.</p>
RANGE	<p>Linear – linear interpolation algorithm is applied, which is very CPU efficient but may be subject to aliasing. Not recommended to use!</p> <p>Low – band-limited interpolation algorithm with low precision is applied.</p> <p>Normal – band-limited interpolation algorithm with normal precision is applied.</p> <p>High – band-limited interpolation algorithm with high precision is applied.</p>
DEFAULT VALUE	Normal

CINEMO_OPTION_AUDIO_REMIX_MOVIEMODE

TYPE	ENUM
DESCRIPTION	<p>Specify whether a 3dB gain should be applied to the center channel. This option applies only when it is required to resample audio before sending to the audio device (see <i>CINEMO_OPTION_AUDIO_DEVICE_SAMPLERATE</i>), and only if the device channel configuration includes a center channel.</p>
RANGE	On, Off
DEFAULT VALUE	On

CINEMO_OPTION_AUDIO_TIMESTRETCH_QUALITY

TYPE	ENUM
DESCRIPTION	Specify the quality of the algorithm used to perform audio time-stretch, for fast or slow forward play speeds.
RANGE	<p>MickyMouse – linear resample algorithm is applied (pitch is not preserved).</p> <p>Fast – fast timestretch algorithm is applied. This mode is recommended for use-cases where the speed is continuously automatically updated and for voice content (e.g. Audiobooks).</p> <p>Best – high quality timestretch algorithm is applied.</p>
DEFAULT VALUE	Best

CINEMO_OPTION_AUDIO_ERROR_FADEIN

TYPE	INTEGER
DESCRIPTION	Duration of time, in milliseconds, to fade in audio when a gap occurs due to an error.
RANGE	0 - 1000
DEFAULT VALUE	200

CINEMO_OPTION_AUDIO_ERROR_FADEOUT

TYPE	INTEGER
DESCRIPTION	Duration of time, in milliseconds, to fade out audio when a gap occurs due to an error.
RANGE	0 - 1000
DEFAULT VALUE	50

CINEMO_OPTION_AUDIO_TRANSITION_FADE

TYPE	INTEGER
DESCRIPTION	Duration, in milliseconds, for audio fade in and fade out during transitions like Start, Pause, Resume or Seek. This option makes these transitions sound softer and prevents clicking artifacts under some circumstances. Please note that the distributed playback delay is used for fade out and in consequence, there still may be clicks, e.g. if the audio buffer is too long in relation to the distributed playback delay.
RANGE	0 - 50
DEFAULT VALUE	10

CINEMO_OPTION_AUDIO_DRCSCALE_CUT

TYPE	INTEGER
DESCRIPTION	Set this value to the desired amount of cut for the high end of dynamic range control for Dolby streams. A value of zero will result in no control for the high end of the range, while a value of 1000 will apply the full high end control as specified in the Dolby bitstream.
RANGE	0 - 1000
DEFAULT VALUE	1000

CINEMO_OPTION_AUDIO_DRCSCALE_BOOST

TYPE	INTEGER
DESCRIPTION	Set this value to the desired amount of boost for the low end of dynamic range control for Dolby streams. Set this value to the desired amount of boost for the low end of dynamic range control for Dolby streams.
RANGE	0 - 1000
DEFAULT VALUE	1000

CINEMO_OPTION_AUDIO_DIALOG_NORMALIZATION

TYPE	BOOLEAN
DESCRIPTION	Set this value to enable dialog normalization for Dolby streams. Dialog normalization adjusts the dialog level in different Dolby streams to a common dialog level for an improved listening experience.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_AUDIO_DEEMPHASIS

TYPE	ENUM
DESCRIPTION	Set this value to enable the deemphasis filter on audio output. When set to On, this option applies to all audio streams. When set to Auto, this option applies only to MPEG audio streams.
RANGE	Off, Auto, On
DEFAULT VALUE	Off

CINEMO_OPTION_AUDIO_AAC_ENABLE_DRC

TYPE	BOOLEAN
DESCRIPTION	Set this value to enable Dynamic Range Control for AAC decoding. When set to TRUE, this option applies to AAC encoded audio streams that contain DRC information.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_AUDIO_SPEEX_ENCODER_CONFIG

TYPE	STRING
DESCRIPTION	Provide a URL parameter string to configure Speex encoder options.
PARAMETER	bitrate – Set the bitrate in kbps (used with disabled VBR).
PARAMETER	abr – Set the average bitrate in kbps (used with enabled VBR).
PARAMETER	quality – Quality used in VBR mode (Range: 0 - 10, Default Value: 4)
PARAMETER	complexity – Complexity used for encoding (trade off between quality and CPU load) (Range: 1 - 10, Default Value: 2)
PARAMETER	wideband – Enable wideband mode (Range: 0 or 1, Default Value: 1)
PARAMETER	vbr – Encode using variable bitrates (VBR) (Range: 0 or 1, Default Value: 0)
PARAMETER	dtx – Enable Speex discontinuous transmission (DTX) which reduces the data rate for periods of silence or stationary background noise (Range: 0 or 1, Default Value: 0)
PARAMETER	plc – Tune the Speex packet loss concealment (PLC) which optimizes encoding for a certain percentage of packet loss. (Range: 0 - 100, Default Value: 0)
EXAMPLE	"?bitrate=40&dtx=1"

CINEMO_OPTION_AUDIO_FLAC_ENCODER_CONFIG

TYPE	STRING
DESCRIPTION	Provide a URL parameter string to configure FLAC encoder options.
PARAMETER	compression_level – Set the libFLAC compression level (Range: 0 - 8, Default Value: 5)
PARAMETER	verify – Enable the libFLAC internal verification (Range: 0 - 1, Default Value: 0)
EXAMPLE	"?compression_level=6&verify=1"

CINEMO_OPTION_AUDIO_AAC_ENCODER_CONFIG

TYPE	STRING
DESCRIPTION	Provide a URL parameter string to configure AAC encoder options.
PARAMETER	granule_length – Set core encoder (AAC) audio frame length in samples (Range: 1024; 512 or 480 for LD/ELD, Default Value: 1024, 512 for LD/ELD)
EXAMPLE	?granule_length=480"

CINEMO_OPTION_AUDIO_DEFAULT_LANGUAGE

TYPE	STRING
DESCRIPTION	Specify the default audio language.
RANGE	ISO 639-1 language code
DEFAULT VALUE	"" (empty)

11.7 Distributed Playback Options

CINEMO_OPTION_DISTRIBUTED_ENABLE

TYPE	BOOLEAN
DESCRIPTION	Enables distributed playback. If enabled, this instance of Cinemo will be able to transmit playback state and content to Cinemo slaves.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_DISTRIBUTED_ENABLE_AUDIO

TYPE	BOOLEAN
DESCRIPTION	Enables the distributed playback transmitting of audio streams.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_DISTRIBUTED_ENABLE_VIDEO

TYPE	BOOLEAN
DESCRIPTION	Enables the distributed playback transmitting of video streams.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_DISTRIBUTED_MIN_TX_PACKETS

CINEMO_OPTION_DISTRIBUTED_MAX_TX_PACKETS

TYPE	INTEGER
DESCRIPTION	Specify the minimum and maximum number of TS packets which may be transmitted by a Cinemo master in one operation. If minimum and maximum are the same, all transfers will be of fixed size.
RANGE	0 - 1024
DEFAULT VALUE	0

CINEMO_OPTION_DISTRIBUTED_MIN_FEEDBACK_PACKETS

CINEMO_OPTION_DISTRIBUTED_MAX_FEEDBACK_PACKETS

TYPE	INTEGER
DESCRIPTION	Specify the minimum and maximum number of TS packets which may be transmitted by a Cinemo slave in one operation. If minimum and maximum are the same, all transfers will be of fixed size.
RANGE	0 - 1024
DEFAULT VALUE	0

CINEMO_OPTION_DISTRIBUTED_MAX_SEND_RATE

TYPE	INTEGER
DESCRIPTION	Specify the maximum data rate (in bytes/s) at which a Cinemo Distributed Playback Server sends samples to a client.
RANGE	0 - 25000000 (0 disables the limit)
DEFAULT VALUE	0

CINEMO_OPTION_DISTRIBUTED_DELAY_MS

TYPE	INTEGER
DESCRIPTION	Specify the time, in milliseconds, to delay commands sent by this master to all distributed playback clients in order to synchronize commands. For example, when playback is paused, the command to resume must be delayed by a certain time on the master, so that it can be executed simultaneously on all clients.
RANGE	0 - 1000
DEFAULT VALUE	125

CINEMO_OPTION_DISTRIBUTED_IP_DSCL

TYPE	INTEGER
DESCRIPTION	Specify the DSCL field value (6bit DSCL value, not including ECN) used for distributed playback master stream traffic. This option overrides CINEMO_OPTION_IP_DSCL unless set to -1. To set the DSCL field for the distributed playback client, add the URL option "ip_dscl".
RANGE	-1 - 63 (-1: use CINEMO_OPTION_IP_DSCL)
DEFAULT VALUE	-1

CINEMO_OPTION_DISTRIBUTED_TCP_MAXSEG

TYPE	INTEGER
DESCRIPTION	Specify the TCP_MAXSEG socket option for distributed playback master stream TCP traffic if supported by the operating system. Please note that various restrictions might be applied to the actual value used by the operating system. To set TCP_MAXSEG for the distributed playback client socket, add the URL option "tcp_maxseg".
RANGE	0-65535 (0: the TCP_MAXSEG socket option will not be set)
DEFAULT VALUE	0

CINEMO_OPTION_DISTRIBUTED_CLOCK_SYNC_MS

TYPE	INTEGER
DESCRIPTION	Specify the frequency, in milliseconds, at which beacon packets are transmitted from master to slave. Beacon packets are used by the master to determine the latency of each slave, and used by slaves to synchronize their clocks with the master clock.
RANGE	0 - 3600000
DEFAULT VALUE	5000

CINEMO_OPTION_DISTRIBUTED_CLOCK_SYNC_PORT

TYPE	STRING
DESCRIPTION	UDP port or port range to use for Distributed Playback clock synchronization.
EXAMPLE	<p>"0" – Random port</p> <p>"40000" – Manually assigned single port</p> <p>"40000-40010" – Manually assigned port range</p>
RANGE	"x" or "x-y" where x, y are within 0–65535. A value of zero or NULL will use random ports.
DEFAULT VALUE	NULL

CINEMO_OPTION_DISTRIBUTED_CLOCK_SYNC_IP_DSCP

TYPE	INTEGER
DESCRIPTION	Specify the DSCP field value (6bit DSCP value, not including ECN) used for distributed playback master clock synchronization. This option overrides CINEMO_OPTION_IP_DSCP unless set to -1. To set the DSCP field for the distributed playback client clock synchronization, add the URL option "clock_sync_ip_dscp".
RANGE	-1 - 63 (-1: use CINEMO_OPTION_IP_DSCP)
DEFAULT VALUE	-1

CINEMO_OPTION_DISTRIBUTED_UPNP_DEVICE_PORT

TYPE	INTEGER
DESCRIPTION	Defines the network port used for providing distributed device descriptions.
RANGE	0–65535. A value of zero will use random ports.
DEFAULT VALUE	0

CINEMO_OPTION_DISTRIBUTED_SERVER_NAME

TYPE	STRING
DESCRIPTION	<p>Specify a user-friendly server name. E.g. John's iPad</p> <p>This value can be obtained through VFS metapool by looking up <code>CINEMO_METANAME_VFS_UPNP_DISTRIBUTED_SERVER_NAME</code>.</p>
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_DISTRIBUTED_SERVER_URL

TYPE	STRING
DESCRIPTION	<p>Specify the distributed playback URL.</p> <p>Depending on the specified protocol, a Cinemo master will either broadcast to, or listen for connections on, the given URL.</p>
Examples:	
	“tcp://:@6050” - TCP server on port 6050.
	“tcp://:@” - TCP server on any port.
	“udp://192.168.1.255:6050” - UDP broadcast to port 6050.
RANGE	UTF-8
DEFAULT VALUE	“tcp://:@”

CINEMO_OPTION_DISTRIBUTED_SSDP_ENABLE_DISCOVERY

TYPE	BOOLEAN
DESCRIPTION	Enables distributed playback discovery. If enabled, this instance of Cinemo will be discoverable by Cinemo slaves.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_DISTRIBUTED_SSDP_NIC

TYPE	STRING
DESCRIPTION	Name of network interface to use for distributed playback.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_DISTRIBUTED_SSDP_MULTICAST_URL

TYPE	STRING
DESCRIPTION	Defines the multicast group ip and port for Distributed Playback SSDP announcement and search for IPv4.
RANGE	UTF-8
DEFAULT VALUE	239.255.255.250:1900

CINEMO_OPTION_DISTRIBUTED_SSDP_MULTICAST_URL_IPV6

TYPE	STRING
DESCRIPTION	Defines the multicast group ip and port for Distributed Playback SSDP announcement and search for IPv6.
RANGE	UTF-8
DEFAULT VALUE	[FF02::C]:1900

CINEMO_OPTION_DISTRIBUTED_SSDP_MAX_AGE

TYPE	INTEGER
DESCRIPTION	Specify an interval after which a Distributed Playback service is considered expired. The unit is seconds.
RANGE	0 - INFINITE.
DEFAULT VALUE	30

11.8 Plugin Options

CINEMO_OPTION_PLUGIN_VIDEO

TYPE	STRING
DESCRIPTION	Specify video plug-in module used by Cinemo. This option should always remain at its default value.
RANGE	"NmeVideo"
DEFAULT VALUE	"NmeVideo"

CINEMO_OPTION_PLUGIN_AUDIO

TYPE	STRING
DESCRIPTION	Specify audio plug-in module used by Cinemo. This option should always remain at its default value.
RANGE	"NmeAudio"
DEFAULT VALUE	"NmeAudio"

CINEMO_OPTION_PLUGIN_SUBPICTURE

TYPE	STRING
DESCRIPTION	Specify subtitle plug-in module used by Cinemo. This option should always remain at its default value.
RANGE	"NmeSubtitle"
DEFAULT VALUE	"NmeSubtitle"

CINEMO_OPTION_PLUGIN_VIDEO_DEVICE

TYPE	STRING
DESCRIPTION	Specify video device plug-in module used by Cinemo. An appropriate platform-specific module is provided with the release and will be set as the default value.
RANGE	<platform-specific module> Supplied as the default video device plug-in module. "NmeDeviceNullVideo" Discard video frames without sending them to a video device.
DEFAULT VALUE	<platform-specific module>

CINEMO_OPTION_PLUGIN_VIDEO_RENDERER

TYPE	STRING
DESCRIPTION	Specify plug-in module used by Cinemo for video rendering.
RANGE	<p>“NmeVmr” Default video renderer.</p> <p>“NmeVmrNull” Discard video frames without waiting for timestamps. No video device will be used.</p>
DEFAULT VALUE	“NmeVmr”

CINEMO_OPTION_PLUGIN_VIDEO_PROCESSOR

TYPE	STRING
DESCRIPTION	Specify optional plug-in module used by Cinemo for video processing. If needed an appropriate platform-specific module is provided with the release and will be set as the default value.
RANGE	<p><platform-specific module> If needed supplied as the default video processor plug-in module.</p>
DEFAULT VALUE	NULL

CINEMO_OPTION_PLUGIN_VIDEO_SOURCE

TYPE	STRING
DESCRIPTION	Specify video source plug-in module used by Cinemo. An appropriate platform-specific module is provided with the release and will be set as the default value.
RANGE	<p><platform-specific module> Supplied as the default video source plug-in module.</p> <p>“NmeVideoSourceNull” Generate black video frames.</p>
DEFAULT VALUE	<platform-specific module>

CINEMO_OPTION_PLUGIN_AUDIO_DEVICE

TYPE	STRING
DESCRIPTION	Specify audio device plug-in module used by Cinemo. An appropriate platform-specific module is provided with the release and will be set as the default value.
RANGE	<p><platform-specific module> Supplied as the default audio device plug-in module.</p> <p>“NmeDeviceNullAudio” Discard audio frames without sending them to an audio device.</p>
DEFAULT VALUE	<platform-specific module>

CINEMO_OPTION_PLUGIN_AUDIO_RENDERER

TYPE	STRING
DESCRIPTION	Specify plug-in module used by Cinemo for audio rendering.
RANGE	<p>"NmeAudioRenderer" Default audio renderer.</p> <p>"NmeNullAudioRenderer" Discard audio frames without waiting for timestamps. No audio device will be used.</p>
DEFAULT VALUE	"NmeAudioRenderer"

CINEMO_OPTION_PLUGIN_AUDIO_MIXER

TYPE	STRING
DESCRIPTION	Specify plug-in module used by Cinemo for audio mixing (Blu-ray).
RANGE	"NmeAudioMixer"
DEFAULT VALUE	"NmeAudioMixer"

CINEMO_OPTION_PLUGIN_AUDIO_CAPTURE

TYPE	STRING
DESCRIPTION	Specify audio capture plug-in module used by Cinemo. An appropriate platform-specific module is provided with the release and will be set as the default value.
RANGE	<p><platform-specific module> Supplied as the default audio device plug-in module.</p> <p>"NmeAudioCaptureNull" Generate silent audio frames.</p>
DEFAULT VALUE	<platform-specific module>

CINEMO_OPTION_PLUGIN_AUDIO_SOURCE

TYPE	STRING
DESCRIPTION	Specify audio source plug-in module used by Cinemo. An appropriate platform-specific module is provided with the release and will be set as the default value.
RANGE	<p><platform-specific module> Supplied as the default audio source plug-in module.</p> <p>"Nme AudioSource" Uses the platform specified audio capture plug-in and device.</p>
DEFAULT VALUE	<platform-specific module>

CINEMO_OPTION_PLUGIN_UNLOAD_DELAY

TYPE	INTEGER
DESCRIPTION	Specify the delay, in seconds, after which an unused plug-in library will be unloaded from memory. This will free system resources, but Cinemo will run slower if the plug-in has to be repeatedly reloaded.
RANGE	0 - 3600. A zero value means that automatic plug-in unloading is disabled.
DEFAULT VALUE	0

11.9 ATAPI Options

CINEMO_OPTION_CD_PLAYBACK_DEVICE_TIMEOUT

CINEMO_OPTION_CD_RIP_DEVICE_TIMEOUT

CINEMO_OPTION_DVD_PLAYBACK_DEVICE_TIMEOUT

CINEMO_OPTION_BD_PLAYBACK_DEVICE_TIMEOUT

TYPE	INTEGER
DESCRIPTION	Specify the ATAPI device timeout value in seconds for each of three cases - CD playback, CD ripping and DVD playback. Note: if an ATAPI read does not complete within the specified time, the read will be aborted and Cinemo robustness algorithms will be invoked. In the case of playback, the disc sector will be skipped and the playback position will advance. In the case of ripping, playback will stop.
RANGE	0 - 60
DEFAULT VALUE	CD playback: 4 CD rip: 20 DVD playback: 4 BD playback: 4

CINEMO_OPTION_CD_PLAYBACK_DEVICE_RETRIES

CINEMO_OPTION_CD_RIP_DEVICE_RETRIES

CINEMO_OPTION_DVD_PLAYBACK_DEVICE_RETRIES

CINEMO_OPTION_BD_PLAYBACK_DEVICE_RETRIES

TYPE	INTEGER
DESCRIPTION	Specify the number of ATAPI device retries. The value here is passed directly to the ATAPI device using MMC command 0x55. Note, not all ATAPI devices support a configurable number of retries.
RANGE	0 - 99
DEFAULT VALUE	CD playback: 0 CD rip: 15 DVD playback: 15 BD playback: 15

CINEMO_OPTION_CD_PLAYBACK_READ_SIZE**CINEMO_OPTION_CD_RIP_READ_SIZE****CINEMO_OPTION_DVD_PLAYBACK_READ_SIZE**

TYPE	INTEGER
DESCRIPTION	Specify the number of sectors in a single ATAPI read operation. Note, a large value here may result in higher transfer speed or better efficiency from an ATAPI device, but will increase the likelihood of read errors if, for example, a CD-ROM is dirty or scratched.
RANGE	1 - 75
DEFAULT VALUE	CD playback: 16 CD rip: 16 DVD playback: 16

CINEMO_OPTION_CD_PLAYBACK_SKIP_ON_ERROR**CINEMO_OPTION_CD_RIP_SKIP_ON_ERROR**

TYPE	INTEGER
DESCRIPTION	Specify the number of sectors to skip after a read error occurs. Note, if another read error occurs after skipping the specified number of sectors, the skip size will be doubled, and so on, until the end of the track is reached. A zero value disables sector skipping and forces playback to always stop with an error.
RANGE	0 - 4500
DEFAULT VALUE	CD playback: 75 CD rip: 0

CINEMO_OPTION_CD_PLAYBACK_READ_SUBCHANNEL

TYPE	BOOL
DESCRIPTION	Specify whether subchannel Q data shall be read from CD during playback. Enable this option if CINEMO_EC_PREGAP events are desired and the CD drive supports subchannel reading.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_DVD_PLAYBACK_SKIP_ON_ERROR

TYPE	INTEGER
DESCRIPTION	Specify the number of seconds to skip, after a read error occurs. Note, if another read error occurs after skipping the specified number of seconds, the skip size will be doubled, and so on, until the end of the track is reached. A zero value disables skipping and forces playback to always stop with an error.
RANGE	0 - 4500
DEFAULT VALUE	1

CINEMO_OPTION_DVD_PLAYBACK_MAX_READ_ERRORS

TYPE	INTEGER
DESCRIPTION	Specify the maximum number of consecutive read errors allowed for DVD playback.
RANGE	0 - 10000
DEFAULT VALUE	0

CINEMO_OPTION_BD_PLAYBACK_FILE_RETRIES

TYPE	INTEGER
DESCRIPTION	Specify the number of software level retries for critical binary file access.
RANGE	0 - 99
DEFAULT VALUE	4

CINEMO_OPTION_BD_PLAYBACK_STREAMING_RETRIES

TYPE	INTEGER
DESCRIPTION	Specify the number of software level retries for non-critical audio/video streaming access.
RANGE	0 - 99
DEFAULT VALUE	0

11.10 DVD Options

CINEMO_OPTION_DVD_CSS_KEY

TYPE	STRING
DESCRIPTION	Obfuscated CSS key. If no key is configured or the key is wrong, Cinemo will fall back to using its built-in CSS key.
DEFAULT VALUE	NULL
LEGAL NOTICE	This fall back to the build-in CSS key shall only be used for development and testing purposes. For production, it is mandatory for customers using the Cinemo Media Engine APIs to provide their own CSS keys. Furthermore, Cinemo reserves the right to remove the build-in CSS keys at any time without notice.

CINEMO_OPTION_DVD_AUDIO_LANGUAGE

CINEMO_OPTION_DVD_SUBPICTURE_LANGUAGE

CINEMO_OPTION_DVD_MENU_LANGUAGE

TYPE	STRING
DESCRIPTION	Specify the default DVD audio, subpicture and menu languages.
RANGE	ISO-636 language code
DEFAULT VALUE	"en"

CINEMO_OPTION_DVD_PARENTAL_COUNTRY

TYPE	STRING
DESCRIPTION	Specify the DVD parental country code.
RANGE	ISO3166 Alpha-2 language code
DEFAULT VALUE	"US"

CINEMO_OPTION_DVD_PARENTAL_LEVEL

TYPE	INTEGER
DESCRIPTION	Specify the initial DVD parental level. Note the DVD parental level may be changed at run time by navigation commands on the disc. In this case, an application must respond appropriately to <i>CINEMO_EC_PARENTAL_LEVEL</i> events.
RANGE	1 - 8
DEFAULT VALUE	8

CINEMO_OPTION_DVD_PARENTAL_LEVEL_CHANGE

TYPE	BOOLEAN
DESCRIPTION	Enable temporary DVD parental level changes by navigation commands.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_DVD_ENFORCE_PUOPS

TYPE	BOOLEAN
DESCRIPTION	Enable the enforcement of DVD prohibited operations (DVD P-UOPs). Note this should always be enabled in production software, and only disabled for testing purposes since it may be possible to avoid forced watching of commercials or trailers on starting playback of a DVD. P-UOPs should always be enforced.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_DVD_PLAYER_REGION_CODE

TYPE	INTEGER
DESCRIPTION	Specify the region code of this DVD player.
RANGE	1 - 8
DEFAULT VALUE	2

CINEMO_OPTION_DVD_ENFORCE_REGION_CODE

TYPE	BOOLEAN
DESCRIPTION	Enable the enforcement of DVD region code checking. If enabled, any attempt to open a DVD whose disc region code does not match the configured player region code will fail. If this option is disabled, then Cinemo will not perform region checks, and it is left solely to the drive/disc authentication mechanism whether the disc will be playable or not.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_DVD_LAZY_CHAPTER_SKIP

TYPE	BOOLEAN
DESCRIPTION	Specify if "lazy chapter skip" should be enabled. When enabled, this let Cinemo decide if it should skip back to the beginning of the current chapter or go to previous chapter, depending on the CINEMO_OPTION_DVD_LAZY CHAPTER SKIP_THRESHOLD option configuration and the actual playback time.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_DVD_LAZY_CHAPTER_SKIP_THRESHOLD

TYPE	INTEGER
DESCRIPTION	Specify the maximum playback time in seconds where the player will switch to the previous chapter instead of the beginning of the current one.
RANGE	1 - 10
DEFAULT VALUE	4

11.11 Subtitle Options

CINEMO_OPTION_SUBTITLE_ENABLE

TYPE	BOOLEAN
DESCRIPTION	Enabling subtitle decoding and playback.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_SUBTITLE_AUTOLOAD

TYPE	BOOLEAN
DESCRIPTION	Enable autoloading of external subtitles from files with the same basename as the video file.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_SUBTITLE_FONT_PATH

TYPE	STRING
DESCRIPTION	Specify the true-type font path, for rendering of external subtitles. Note, if this option is not specified, a built-in Unicode font will be used instead. This is sufficient for most purposes.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_SUBTITLE_FONT_COLOR

TYPE	STRING
DESCRIPTION	Specify the color for rendering external subtitle fonts.
RANGE	"R,G,B,A"
DEFAULT VALUE	"255,255,255,255"

CINEMO_OPTION_SUBTITLE_OUTLINE_COLOR

TYPE	STRING
DESCRIPTION	Specify the color for rendering external subtitle font outline.
RANGE	"R,G,B,A"
DEFAULT VALUE	"0,0,0,255"

CINEMO_OPTION_SUBTITLE_FONT_SIZE

TYPE	INTEGER
DESCRIPTION	Specify the font size for rendering external subtitles.
RANGE	5 - 200
DEFAULT VALUE	20

CINEMO_OPTION_SUBTITLE_OUTLINE_WIDTH

TYPE	INTEGER
DESCRIPTION	Specify the outline width for rendering external subtitles.
RANGE	0 - 10
DEFAULT VALUE	2

CINEMO_OPTION_SUBTITLE_ITALIC**CINEMO_OPTION_SUBTITLE_BOLD****CINEMO_OPTION_SUBTITLE_UNDERLINE**

TYPE	BOOLEAN
DESCRIPTION	Flags for rendering external subtitles.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_SUBTITLE_ALIGNHORZ

TYPE	ENUM
DESCRIPTION	Specify horizontal alignment for rendering external subtitles.
RANGE	"Off", "Left", "Center", "Right"
DEFAULT VALUE	"Center"

CINEMO_OPTION_SUBTITLE_ALIGNVERT

TYPE	ENUM
DESCRIPTION	Specify vertical alignment for rendering external subtitles.
RANGE	"Off", "Top", "Center", "Bottom"
DEFAULT VALUE	"Bottom"

CINEMO_OPTION_SUBTITLE_FADEIN**CINEMO_OPTION_SUBTITLE_FADEOUT**

TYPE	INTEGER
DESCRIPTION	Specify the time in milliseconds for external subtitle fade in and out effects.
RANGE	0 - 2000. A zero value will disable the fade effect.
DEFAULT VALUE	0

CINEMO_OPTION_SUBTITLE_IDXSUB

TYPE	BOOLEAN
DESCRIPTION	Specify whether IDX subtitle formatting is enabled.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_SUBTITLE_WIDTH

TYPE	INTEGER
DESCRIPTION	Specify the width, in pixels, of the external subtitle overlay surface.
RANGE	320 - 1920
DEFAULT VALUE	800

CINEMO_OPTION_SUBTITLE_HEIGHT

TYPE	INTEGER
DESCRIPTION	Specify the height, in pixels, of the external subtitle overlay surface.
RANGE	240 - 1440
DEFAULT VALUE	450

CINEMO_OPTION_SUBTITLE_VIEWPORT_ALIGNED

TYPE	BOOLEAN
DESCRIPTION	Specify whether subtitle is aligned to whole viewport instead of video area. This feature to show subtitle outside of video area is not support by all video devices.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_SUBTITLE_DEFAULT_CHARSET

TYPE	STRING
DESCRIPTION	Specify a comma separated list of file encodings for external subtitle files that don't have a UTF-8 or UTF-16 BOM. The respective encoding is used. Otherwise the encoding from the list which yields the least amount of unrecognizable characters is finally chosen.
RANGE	utf-8, utf-16-le, utf-16-be, utf-32-le, utf-32-be, iso-646, iso-8859-1, iso-8859-2, iso-8859-3, iso-8859-4, iso-8859-5, iso-8859-6, iso-8859-7, iso-8859-8, iso-8859-9, iso-8859-10, iso-8859-11, iso-8859-13, iso-8859-14, iso-8859-15, iso-8859-16, cp-037, cp-437, cp-500, cp-720, cp-737, cp-775, cp-850, cp-852, cp-855, cp-857, cp-858, cp-860, cp-861, cp-862, cp-863, cp-864, cp-865, cp-866, cp-869, cp-874, cp-875, cp-1026, cp-1250, cp-1251, cp-1252, cp-1253, cp-1254, cp-1255, cp-1256, cp-1257, cp-1258, cp-932, cp-936, cp-949, cp-950, euc-jp, euc-kr, koi8-r, koi8-u, gb-18030, tis-620, iso-2022-cn, iso-2022-kr, hkscs, gb-2312
DEFAULT VALUE	NONE

CINEMO_OPTION_SUBTITLE_DELAY_MS

TYPE	INTEGER
DESCRIPTION	Specify the time, in milliseconds, to delay subtitles in order to synchronize with audio for this playback client.
RANGE	-100,000 to 100,000
DEFAULT VALUE	0

11.12 Thread Priority Options

CINEMO_OPTION_PRIO_PREFETCH

TYPE	INTEGER
DESCRIPTION	Specify the thread priority for threads concerned with stream prefetch. Normally this value should be higher than normal to avoid data loss. This value should only be changed after consultation with Cinemo.
RANGE	Platform specific
DEFAULT VALUE	Platform specific

CINEMO_OPTION_PRIO_PLAYBACK

TYPE	INTEGER
DESCRIPTION	Specify the thread priority for playback threads. This value should only be changed after consultation with Cinemo.
RANGE	Platform specific.
DEFAULT VALUE	Platform specific.

CINEMO_OPTION_PRIO_ENCODING

TYPE	INTEGER
DESCRIPTION	Specify the thread priority for encoding threads. Normally this value should be lower than normal, since encoding is typically a background activity. This value should only be changed after consultation with Cinemo.
RANGE	Platform specific.
DEFAULT VALUE	Platform specific.

CINEMO_OPTION_PRIO_AUDIO_DEVICE

CINEMO_OPTION_PRIO_VIDEO_DEVICE

TYPE	INTEGER
DESCRIPTION	Specify the thread priority for audio and video rendering devices. Normally these values should be the highest priority, since dropping PCM samples or video frames in the renderer results in ugly artifacts and a poor user experience. This value should only be changed after consultation with Cinemo.
RANGE	Platform specific.
DEFAULT VALUE	Platform specific.

CINEMO_OPTION_PRIO_GRAPHICS

TYPE	INTEGER
DESCRIPTION	Specify the thread priority for graphics menus and BDJ application. Normally this value should be lower than normal, since video and audio playback should not suffer from heavy graphics animations. This value should only be changed after consultation with Cinemo.
RANGE	Platform specific.
DEFAULT VALUE	Platform specific.

CINEMO_OPTION_PRIO_INDEXING

TYPE	INTEGER
DESCRIPTION	Specify the thread priority for MM indexing threads. Normally this value should be lower than normal, since indexing is typically a background activity. This value should only be changed after consultation with Cinemo.
RANGE	Platform specific.
DEFAULT VALUE	Platform specific.

CINEMO_OPTION_PRIO_IAP_LINK

TYPE	INTEGER
DESCRIPTION	Specify the thread priority for iAP2 link send and receive threads. Normally this value should be higher than normal to be able to fulfill protocol timing requirements. This value should only be changed after consultation with Cinemo.
RANGE	Platform specific.
DEFAULT VALUE	Platform specific.

11.13 Thread Scheduling Policy Options

CINEMO_OPTION_SCHED_POLICY_PREFETCH

TYPE	ENUM
DESCRIPTION	Specify the schedule policy for threads concerned with stream prefetch. This value should only be changed after consultation with Cinemo.
RANGE	“Other”, “FIFO”, “RR”, “Batch”, “Unspecified”, “Idle”, “Nice”
DEFAULT VALUE	“Unspecified”

CINEMO_OPTION_SCHED_POLICY_PLAYBACK

TYPE	ENUM
DESCRIPTION	Specify the schedule policy for playback threads. This value should only be changed after consultation with Cinemo.
RANGE	“Other”, “FIFO”, “RR”, “Batch”, “Unspecified”, “Idle”, “Nice”
DEFAULT VALUE	“Unspecified”

CINEMO_OPTION_SCHED_POLICY_ENCODING

TYPE	ENUM
DESCRIPTION	Specify the schedule policy for encoding threads. This value should only be changed after consultation with Cinemo.
RANGE	“Other”, “FIFO”, “RR”, “Batch”, “Unspecified”, “Idle”, “Nice”
DEFAULT VALUE	“Unspecified”

CINEMO_OPTION_SCHED_POLICY_AUDIO_DEVICE

CINEMO_OPTION_SCHED_POLICY_VIDEO_DEVICE

TYPE	ENUM
DESCRIPTION	Specify the schedule policy for audio and video rendering devices. This value should only be changed after consultation with Cinemo.
RANGE	“Other”, “FIFO”, “RR”, “Batch”, “Unspecified”, “Idle”, “Nice”
DEFAULT VALUE	“Unspecified”

CINEMO_OPTION_SCHED_POLICY_GRAPHICS

TYPE	ENUM
DESCRIPTION	Specify the schedule policy for graphics menus and BDJ application. This value should only be changed after consultation with Cinemo.
RANGE	“Other”, “FIFO”, “RR”, “Batch”, “Unspecified”, “Idle”, “Nice”
DEFAULT VALUE	“Unspecified”

CINEMO_OPTION_SCHED_POLICY_INDEXING

TYPE	ENUM
DESCRIPTION	Specify the schedule policy for MM indexing threads. This value should only be changed after consultation with Cinemo.
RANGE	"Other", "FIFO", "RR", "Batch", "Unspecified", "Idle", "Nice"
DEFAULT VALUE	"Unspecified"

11.14 Media Management Options

CINEMO_OPTION_MM_ENABLE

TYPE	BOOLEAN
DESCRIPTION	Enable MM in Cinemo Demo Application.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_MM_ENABLE_JOURNALING

TYPE	BOOLEAN
DESCRIPTION	Enable Journaling functionality of MM. Changing this option will have no effect on already existing volumes.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_SERVER_NAME

TYPE	STRING
DESCRIPTION	Specify the name of the MM server. This is also the value of the friendlyName field in the UPnP device description of the MM server.
RANGE	UTF-8
DEFAULT VALUE	"Cinemo Media Server"

CINEMO_OPTION_MM_SERVER_UPNP_MODEL_NAME

TYPE	STRING
DESCRIPTION	Specify the value of the modelName field in the UPnP device description of the MM server.
RANGE	UTF-8
DEFAULT VALUE	"Cinemo Media Server"

CINEMO_OPTION_MM_SERVER_UPNP_MODEL_DESCRIPTION

TYPE	STRING
DESCRIPTION	Specify the value of the modelDescription field in the UPnP device description of the MM server.
RANGE	UTF-8
DEFAULT VALUE	"Cinemo Media Server"

CINEMO_OPTION_MM_SERVER_UPNP_MANUFACTURER

TYPE	STRING
DESCRIPTION	Specify the value of the manufacturer field in the UPnP device description of the MM server.
RANGE	UTF-8
DEFAULT VALUE	"Cinemo GmbH"

CINEMO_OPTION_MM_SERVER_UPNP_MANUFACTURER_URL

TYPE	STRING
DESCRIPTION	Specify the value of the manufacturerURL field in the UPnP device description of the MM server.
RANGE	UTF-8
DEFAULT VALUE	"http://www.cinemo.com"

CINEMO_OPTION_MM_SERVER_UUID

TYPE	STRING
DESCRIPTION	Specify the 128-bit UPNP UUID of the MM server. UUID must be of the form "01234567-89AB-CDEF-0123-456789ABCDEF". If unspecified, a random UUID will be generated each time the MM server is run. The DCE 1.1 algorithm is recommended to generate such UUID. Typically applications will generate a UUID for this option and store it for reuse at the next start up so clients will be able to recognize previously known servers.
RANGE	UTF-8
DEFAULT VALUE	NULL (unspecified)

CINEMO_OPTION_MM_SERVER_ICON_24BITS

TYPE	STRING
DESCRIPTION	URL of the 24-bit icon for the Media Management server
RANGE	UTF-8
DEFAULT VALUE	"res://cinemo_icon_256x256.png"

CINEMO_OPTION_MM_SERVER_ICON_32BITS

TYPE	STRING
DESCRIPTION	URL of the 32-bit icon for the Media Management server
RANGE	UTF-8
DEFAULT VALUE	"res://cinemo_icon_256x256_transparent.png"

CINEMO_OPTION_MM_SERVER_URL

TYPE	STRING
DESCRIPTION	Specify a fixed TCP/IP Address and Port for the MM HTTP server.
Examples:	
“@” or “@:0” - Any IPv4 and IPv6 Address (Dualstack), Port 0	
“0” or “0.0.0.0:0” or “0:0” - Any IPv4 Address, Port 0	
“[::]” or “[::]:0” - Any IPv6 Address, Port 0	
“@:44444” - Any IPv4 and IPv6 Address, Port 44444	
“0:44444” - Any IPv4 Address, Port 44444	
“[::]:44444” - Any IPv6 Address, Port 44444	
“127.0.0.1” - Localhost IPv4	
“[::1]” - Localhost IPv6	
“192.168.1.42:33333” - fully specified IPv4 Address and Port	
RANGE	UTF-8
DEFAULT VALUE	“@”

CINEMO_OPTION_MM_SSDP_ENABLE_DISCOVERY

TYPE	BOOLEAN
DESCRIPTION	Enables media management discovery. If enabled, this instance of Cinemo MM will be discoverable by UPNP clients.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_MM_SSDP_NIC

TYPE	STRING
DESCRIPTION	Name of the network interface for MM SSDP
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_SSDP_MULTICAST_URL

TYPE	STRING
DESCRIPTION	Defines the multicast group ip and port for Media Management SSDP announcement and search for IPv4.
RANGE	UTF-8
DEFAULT VALUE	239.255.255.250:1900

CINEMO_OPTION_MM_SSDP_MULTICAST_URL_IPV6

TYPE	STRING
DESCRIPTION	Defines the multicast group ip and port for Media Management SSDP announcement and search for IPv6.
RANGE	UTF-8
DEFAULT VALUE	[FF02::C]:1900

CINEMO_OPTION_MM_SSDP_MAX_AGE

TYPE	INTEGER
DESCRIPTION	Specify the interval in seconds after which the MM service is considered expired.
RANGE	0 - INFINITE.
DEFAULT VALUE	60

CINEMO_OPTION_MM_DDP_QUERY_SERVER_URL

TYPE	STRING
DESCRIPTION	Specify the URL of the DDP query server created by Media Management if either a query or a metadata expression handler is registered. Example: ddp://127.0.0.1:30002. By default the server will be created on all interfaces.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_DDP_SERVER_URL

TYPE	STRING
DESCRIPTION	Specify the URL of the DDP server automatically created by Media Management. Example: ddp://127.0.0.1:30001. By default the server will be created on all interfaces.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_CONFIG_XML

TYPE	STRING
DESCRIPTION	Specify the path to Cinemo MM Configuration file.
RANGE	UTF-8
DEFAULT VALUE	"res://cinemo_mm.xml"

CINEMO_OPTION_MM_CONFIG_UCA_DUCET_TABLE

TYPE	STRING
DESCRIPTION	Specify the location of the file containing the "Default Unicode Collation Element Table" (DUCET).
RANGE	UTF-8
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_LOCALES_CONFIG_XML

TYPE	STRING
DESCRIPTION	Specify the path to Cinemo MM Locales file. If set to NULL, the locale is configured via the options <code>CINEMO_OPTION_MM_ICU_SORT_*</code> and <code>CINEMO_OPTION_MM_ICU_SEARCH_*</code> .
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_LOCALES_DEFAULT

TYPE	STRING
DESCRIPTION	Default locale to use if no locale is specified in the query. Requires a valid Cinemo MM Locales file.
RANGE	UTF-8
DEFAULT VALUE	en

CINEMO_OPTION_MM_LOCALES_AVAILABLE

TYPE	STRING
DESCRIPTION	Comma separated list of locales that are available for clients. For these locales, a sortkey will be extracted for efficient sorting and comparison. Requires a valid Cinemo MM Locales file.
RANGE	UTF-8
DEFAULT VALUE	en

CINEMO_OPTION_MM_LOCALES_COLLATION_FALLBACK

TYPE	BOOLEAN
DESCRIPTION	Whether to use collation fallback for locales, if no sortkey is available.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_ICU_MODE

TYPE	ENUM
DESCRIPTION	Set ICU mode. Only used if <i>CINEMO_OPTION_MM_LOCALES_CONFIG_XML</i> is NULL.
RANGE	Disable: Do not use ICU Sort: Use ICU for sorting SortSearch: Use ICU for sorting and searching
DEFAULT VALUE	Sort

CINEMO_OPTION_MM_ICU_SORT_LOCALE

TYPE	STRING
DESCRIPTION	Locale that should be used for sorting (and index generation). Only used if <i>CINEMO_OPTION_MM_LOCALES_CONFIG_XML</i> is NULL.
RANGE	UTF-8
DEFAULT VALUE	"en"

CINEMO_OPTION_MM_ICU_SORT_NUMERIC

TYPE	INTEGER
DESCRIPTION	Numeric sorting will be used. If activated 2 will sort before 100. A value of -1 means, don't set at all and use ICU default. Only used if <i>CINEMO_OPTION_MM_LOCALES_CONFIG_XML</i> is NULL.
RANGE	-1,0,1
DEFAULT VALUE	1

CINEMO_OPTION_MM_ICU_SORT_IGNORE

TYPE	INTEGER
DESCRIPTION	Ignores variable characters on the UCA: white space, punctuation marks, and symbols. If activated these characters will be ignored on sorting. A value of -1 means, don't set at all and use ICU default. Only used if <i>CINEMO_OPTION_MM_LOCALES_CONFIG_XML</i> is NULL.
RANGE	-1,0,1
DEFAULT VALUE	1

CINEMO_OPTION_MM_ICU_SEARCH_LOCALE

TYPE	STRING
DESCRIPTION	Locale that should be used for searching. Caution: the base locale should be the same as for sorting, however the locale options may be different. Only used if CINEMO_OPTION_MM_LOCALES_CONFIG_XML is NULL.
RANGE	UTF-8
DEFAULT VALUE	"en"

CINEMO_OPTION_MM_ICU_SEARCH_IGNORE

TYPE	INTEGER
DESCRIPTION	Ignores variable characters on the UCA: white space, punctuation marks, and symbols. If activated these characters will be ignored on sorting. A value of -1 means, don't set at all and use ICU default. Caution: When this option is "1", searching for ignored characters will not be possible (e.g. "-"). Only used if CINEMO_OPTION_MM_LOCALES_CONFIG_XML is NULL.
RANGE	-1,0,1
DEFAULT VALUE	-1

CINEMO_OPTION_MM_ICU_DIR

TYPE	STRING
DESCRIPTION	Specify the directory where ICU libraries are located. The library names are expected to be <ul style="list-style-type: none"> • Unix-like: libicudata.so.XX, libicuuc.so.XX and libicui18n.so.XX • Windows: icudtXX.dll, icuucXX.dll and icuinXX.dll where XX is the version of the installed ICU library. If no directory is specified, an ICU library will be searched using the dynamic library loader.
RANGE	UTF-8

CINEMO_OPTION_MM_ICU_VERSION

TYPE	INTEGER
DESCRIPTION	Specify the ICU library version. If 0 specified the SDK automatically will use the newest version in the directory specified by CINEMO_OPTION_MM_SORT_ICU_DIR . If the directory is not specified it tries to find the newest version using the dynamic library loader.
RANGE	0,46,48,49,50,51,52,53,54,55,56,57,58,59,60
DEFAULT VALUE	0

CINEMO_OPTION_MM_ICU_DATAFILE

TYPE	STRING
DESCRIPTION	Specify the complete path of the ICU data file. The file can be customized using ICU command line tools or online (http://apps.icu-project.org/datacustom/). Note: The data of this file must match the ICU library version. Typical name: <i>icudt55l.dat</i> .
RANGE	UTF-8

CINEMO_OPTION_MM_ICU_DATAFILE_MMAP

TYPE	BOOLEAN
DESCRIPTION	Use mmap() instead of loading the ICU data file completely into allocated memory. This might save some memory at the cost of slight performance decrease.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_MOUNT_PATH

TYPE	STRING
DESCRIPTION	Specify a path to automatically mount when MM server is started. The special path value “~” corresponds to the user’s home folder.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_MOUNT_EXISTING_VOLUMES

TYPE	BOOLEAN
DESCRIPTION	Enables the automatic mounting, on MM start-up, of all persistent volumes which have been indexed before. If not set, persistent volumes will be restored in a dismounted state. This only affects the start-up behaviour of the MM server. Note that this option will mount volumes under their previous mount path. Consequently, if you are using IAP, MTP, or MSD volumes where the mountpath can change, FALSE is recommended.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_MOUNT_USE_UUID

TYPE	BOOLEAN
DESCRIPTION	Specify whether or not to automatically generate UUID for mounted volumes.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_MM_WATCH_ENABLE

TYPE	BOOLEAN
DESCRIPTION	Enables watching for insertion and removal of media devices. Please note, that this feature is only meant for testing purposes. It is strongly advised to implement a BSP dependant watching for volume inserting and removal around ICinemoMM::AddIndexedVolume() . If set, device insertion and removal will be automatically detected by Cinemo MM, indexed volumes will be created for inserted devices, and indexing will start and stop automatically.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_WATCH_PATH

TYPE	STRING
DESCRIPTION	Specifies the path to watch for removable media. If NULL, which is recommended, then Cinemo MM will instead watch /proc/mounts on the target system.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_WATCH_EXCLUDE

TYPE	STRING
DESCRIPTION	A regular expression to specify which paths which should be excluded from the watch for removable media.
RANGE	Regular Expression
DEFAULT VALUE	"*(/dev/cd /dev/dvd /dev/scd /dev/sr /boot).*"

CINEMO_OPTION_MM_WATCH_OPTICAL_DRIVES

TYPE	BOOLEAN
DESCRIPTION	Specify whether or not to watch optical drives.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_WATCH_PARENT_ID

TYPE	STRING
DESCRIPTION	Specify the ID of the parent volume group in which to add indexed volumes created by the watch process.
RANGE	UTF-8
DEFAULT VALUE	"1" - corresponding to volume group '1' in default Cinemo MM Configuration .

CINEMO_OPTION_MM_WATCH_VOLUME_NAME

TYPE	STRING
DESCRIPTION	Specify the name of indexed volumes created by the watch process.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_WATCH_VOLUME_TYPE

TYPE	STRING
DESCRIPTION	Specify the type of indexed volumes created by the watch process.
RANGE	UTF-8
DEFAULT VALUE	“Auto-mounted volume”

CINEMO_OPTION_MM_WATCH_VOLUME_HIERARCHY

TYPE	STRING
DESCRIPTION	Specify that indexed volumes created by the watch process should be created as a child of a new volume group, where a new volume group is created for each volume, and that each volume group should expose a hierarchy defined by this identifier in the Cinemo MM Configuration .
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_FILENAME_WHITELIST

TYPE	STRING
DESCRIPTION	Specify a regular expression that defines a filename whitelist for indexing.
RANGE	Regular expression
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_FILENAME_EXCLUDE

TYPE	STRING
DESCRIPTION	Specify which filenames should be excluded from indexing.
RANGE	Regular expression
DEFAULT VALUE	“\\.(?i)(?:db)\$”

CINEMO_OPTION_MM_FILENAME_EXCLUDE_FULLPATH (DEPRECATED)

TYPE	STRING
DESCRIPTION	Specify which volume paths should be excluded from indexing. DEPRECATED: for performance reasons use the options CINEMO_OPTION_MM_FILENAME_EXCLUDE_FULLPATH_FILE and CINEMO_OPTION_MM_FILENAME_EXCLUDE_FULLPATH_FOLDER. Note that CINEMO_OPTION_MM_FILENAME_EXCLUDE_FULLPATH_FOLDER does not expect a separator at the end of a folder name.
RANGE	Regular expression
DEFAULT VALUE	""

CINEMO_OPTION_MM_FILENAME_EXCLUDE_FULLPATH_FILE

TYPE	STRING
DESCRIPTION	Specify which files shall be excluded from indexing by volume path.
RANGE	Regular expression
DEFAULT VALUE	""

CINEMO_OPTION_MM_FILENAME_EXCLUDE_FULLPATH_FOLDER

TYPE	STRING
DESCRIPTION	Specify which folders shall be excluded from indexing by volume path.
RANGE	Regular expression
DEFAULT VALUE	""

CINEMO_OPTION_MM_FILENAME_AUDIO

TYPE	STRING
DESCRIPTION	Specify which filenames will be indexed as audio during the first pass, when only the filename is known and the contents have not yet been analysed.
RANGE	Regular expression
DEFAULT VALUE	All known audio filename extensions

CINEMO_OPTION_MM_FILENAME_AUDIO_BOOKS

TYPE	STRING
DESCRIPTION	Specify which filenames will be indexed as audio books during the first pass, when only the filename is known and the contents have not yet been analysed.
RANGE	Regular expression
DEFAULT VALUE	All known audio book filename extensions

CINEMO_OPTION_MM_FILENAME_VIDEO

TYPE	STRING
DESCRIPTION	Specify which filenames will be indexed as video during the first pass, when only the filename is known and the contents have not yet been analysed.
RANGE	Regular expression
DEFAULT VALUE	All known video filename extensions

CINEMO_OPTION_MM_FILENAME_IMAGE

TYPE	STRING
DESCRIPTION	Specify which filenames will be indexed as images during the first pass, when only the filename is known and the contents have not yet been analysed.
RANGE	Regular expression
DEFAULT VALUE	All known image filename extensions

CINEMO_OPTION_MM_FILENAME_PLAYLIST

TYPE	STRING
DESCRIPTION	Specify which filenames will be indexed as playlists during the first pass, when only the filename is known and the contents have not yet been analysed.
RANGE	Regular expression
DEFAULT VALUE	All known playlist filename extensions

CINEMO_OPTION_MM_FILENAME_COVER_IMAGES

TYPE	STRING
DESCRIPTION	Specify which filenames are considered external cover images on the folder level.
RANGE	Regular expression
DEFAULT VALUE	"^ (?i) folder\\\\. (?:jpg jpeg)\$"

CINEMO_OPTION_MM_GENRE_AUDIO_BOOKS

TYPE	STRING
DESCRIPTION	Specify which genres will be indexed as audio books after metadata extraction.
RANGE	Regular expression
DEFAULT VALUE	"^ (?i) (?:Audiobook Audiobooks Spoken)\$"

CINEMO_OPTION_MM_GENRE_PODCASTS

TYPE	STRING
DESCRIPTION	Specify which genres will be indexed as podcasts after metadata extraction.
RANGE	Regular expression
DEFAULT VALUE	"^ (?i) (?:Podcast) \$"

CINEMO_OPTION_MM_INDEXER_FASTPLAY

TYPE	STRING
DESCRIPTION	Specify comma separated list of UPNPIDs that shall be prioritized by the indexer.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_INDEXER_FOLDERS_PRIORITY

TYPE	STRING
DESCRIPTION	Specify comma separated list of folders to index first in the file indexing phase (applicable to MSD and MTP volumes).
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_INDEXER_FILESYSTEM_HINT

TYPE	ENUM
DESCRIPTION	Specify the file system to be assumed for mass storage devices. The hint is used under linux to retrieve the hidden attributes of files and folders. Hidden attribute extraction is disabled if the hint is "None". Note that setting this option to any other value than "None" can influence the file indexing performance. Whether specific file system attributes are supported depends on the BSP.
RANGE	"None", "Auto", "Linux", "FAT", "NTFS", "HFS+"

CINEMO_OPTION_MM_INDEXER_SKIP_DOT_FILES

TYPE	BOOLEAN
DESCRIPTION	Skip linux style dotfiles during indexing.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_MM_INDEXER_SKIP_DOT_FOLDERS

TYPE	BOOLEAN
DESCRIPTION	Skip linux style dotfolders during indexing.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_MM_INDEXER_SKIP_HIDDEN_FILES

TYPE	BOOLEAN
DESCRIPTION	Skip hidden files during indexing.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_MM_INDEXER_SKIP_HIDDEN_FOLDERS

TYPE	BOOLEAN
DESCRIPTION	Skip hidden folders during indexing.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_MM_INDEXER_SKIP_SYMLINKS

TYPE	BOOLEAN
DESCRIPTION	Skip symbolic links on mass storage devices.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_INDEXER_MINIMUM_FILESIZE

TYPE	INTEGER
DESCRIPTION	Specify the minimum size of files to be considered for indexing in kilobytes (KB) for mass storage device and MTP volumes.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_INDEXER_METADATA_SKIP

TYPE	BOOLEAN
DESCRIPTION	Do not extract metadata from indexing MSD volumes.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_INDEXER_COVERART_SKIP

TYPE	BOOLEAN
DESCRIPTION	Do not extract thumbnails from coverart metadata.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_INDEXER_NOMEDIA_FOLDER_TAGS

TYPE	STRING
DESCRIPTION	Specify a comma separated list of filenames that exclude a folder from indexing on MTP volumes. If a file of this name is inside a folder, the content of the folder will not be considered by media management.
RANGE	UTF-8
DEFAULT VALUE	.nomedia

CINEMO_OPTION_MM_INDEXER_NOMEDIA_FILES_THRESHOLD

TYPE	INTEGER
DESCRIPTION	Specify the maximum number of detected non media files in a folder (based on the filename and the respective exclude patterns) that if reached will cause the indexer to exclude the entire folder.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_INDEXER_CLASS_FROM_FILENAME

TYPE	BOOLEAN
DESCRIPTION	If TRUE, the upnp:class of indexed media items will be solely defined by their filenames according to CINEMO_OPTION_MM_FILENAME_* options. The class will not be changed even if Cinemo detects a different class during metadata extraction.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_INDEXER_QUOTA_MINIMIZE_ACCESS

TYPE	BOOLEAN
DESCRIPTION	If TRUE, indexing is aborted if the tracks per volume quota (CINEMO_OPTION_MM_QUOTA_TRACKS_PER_VOLUME) is reached.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_MM_THUMB_FORMAT

TYPE	ENUM
DESCRIPTION	Specify the image format of thumbnails.
RANGE	"JPEG", "PNG"
DEFAULT VALUE	"JPEG"

CINEMO_OPTION_MM_THUMB_WIDTH**CINEMO_OPTION_MM_THUMB_HEIGHT**

TYPE	INTEGER
DESCRIPTION	Specify the width and height of Media Management thumbnails.
RANGE	0 - INFINITE
DEFAULT VALUE	120

CINEMO_OPTION_MM_THUMB_QUALITY

TYPE	INTEGER
DESCRIPTION	Specify the quality in which to JPEG encode thumbnails.
RANGE	0 - 100
DEFAULT VALUE	75

CINEMO_OPTION_MM_THUMB_SKIP_VIDEOS

TYPE	BOOLEAN
DESCRIPTION	Do not extract thumbnails by decoding video for mass storage devices.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_THUMB_SKIP_IMAGES

TYPE	BOOLEAN
DESCRIPTION	Do not extract thumbnails by decoding images for mass storage devices.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_THUMB_COVERART_LIMIT

TYPE	INTEGER
DESCRIPTION	Limit the number of thumbnails generated from coverart during indexing.
RANGE	0 - INFINITE
DEFAULT VALUE	1

CINEMO_OPTION_MM_THUMB_COVERART_PRIO

TYPE	STRING
DESCRIPTION	Prioritize coverart types during thumbnail generation.
RANGE	"p1, ..., pn" where each entry is either the thumbnail type according to id3v2/flac image types or "*" for any type.
EXAMPLE	<p>"*" – No prioritization, use order of occurrence</p> <p>"3" – Only consider front cover</p> <p>"3,*" – Priorize front cover, otherwise use order of occurrence</p>
DEFAULT VALUE	"3,*"

CINEMO_OPTION_MM_THUMB_DEFAULT_STRATEGY

TYPE	STRING
DESCRIPTION	Set the default strategy for retrieving thumbnails in queries.
RANGE	"p1, ..., pn" where each entry prioritizes a specific thumbnail type for MM queries.
EXAMPLE	<p>"*" – No prioritization, use first thumb stored</p> <p>"3" – Retrieve front cover if present</p> <p>"3,*" – Retrieve front cover if present, otherwise retrieve first thumb stored</p> <p>"*,3" – Priorize any other stored thumb over front cover</p> <p>"A" – Retrieve thumb from album children. (only applied for object.container.album and derived classes)</p> <p>"F" – Retrieve thumb from folder cover image. (only applied for object.item.audiolitem and derived classes)</p>
DEFAULT VALUE	"*,A,F"

CINEMO_OPTION_MM_THUMB_STORAGE_PATH

TYPE	STRING
DESCRIPTION	Specify the path for storing Media Management thumbnails. If not specified, the default behavior is to store thumbnails in the Cinemo database.
RANGE	UTF-8
DEFAULT VALUE	NULL

CINEMO_OPTION_MM_QUOTA_FOLDER_DEPTH

TYPE	INTEGER
DESCRIPTION	Specifies the maximum indexed folder depth, starting from volume mount path. A value of zero means no limitation.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_QUOTA_FOLDERS_PER_VOLUME

TYPE	INTEGER
DESCRIPTION	Specifies the maximum number of folders that may be indexed in a single volume. A value of zero means no limitation.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_QUOTA_TRACKS_PER_FOLDER

TYPE	INTEGER
DESCRIPTION	Specifies the maximum number of files that may be indexed in a single folder. A value of zero means no limitation.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_QUOTA_TRACKS_PER_VOLUME

TYPE	INTEGER
DESCRIPTION	Specifies the maximum number of files that may be indexed in a single volume. A value of zero means no limitation.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_QUOTA_PLAYLISTS_PER_VOLUME

TYPE	INTEGER
DESCRIPTION	Specifies the maximum number of playlists that may be indexed in a single MSD volume. A value of zero means no limitation.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_QUOTA_PLAYLISTTRACKS_PER_VOLUME

TYPE	INTEGER
DESCRIPTION	Specifies the maximum total number of parsed entries in playlists that may be indexed in a single MSD volume. A value of zero means no limitation.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_QUOTA_PLAYLISTTRACKS_PER_PLAYLIST

TYPE	INTEGER
DESCRIPTION	Specifies the maximum number of parsed entries in a playlist that may be indexed from a MSD volume. A value of zero means no limitation.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_QUOTA_RAM

TYPE	INTEGER
DESCRIPTION	Specifies how much RAM in megabytes (MB) can be used by MM.
RANGE	0 - INFINITE
DEFAULT VALUE	64

CINEMO_OPTION_MM_QUOTA_STORAGE

TYPE	INTEGER
DESCRIPTION	Specifies how much disk storage space in megabytes (MB) can be used by MM.
RANGE	0 - INFINITE
DEFAULT VALUE	256

CINEMO_OPTION_MM_JOURNAL_NODE_EVENT_LIMIT

TYPE	INTEGER
DESCRIPTION	Size limit on number of node events. A value of 0 turns node event limitation off.
RANGE	0 - INFINITE
DEFAULT VALUE	0

CINEMO_OPTION_MM_HIERARCHY_CASE_SENSITIVE

TYPE	BOOLEAN
DESCRIPTION	Hierarchy node generation with case sensitivity.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_MM_SQLITE_SYNCHRONOUS

TYPE	ENUM
DESCRIPTION	Sets the sqlite synchronous mode. Corresponds to the sqlite pragma synchronous.
RANGE	“OFF”, “NORMAL”, “FULL”
DEFAULT VALUE	“OFF”

CINEMO_OPTION_MM_SQLITE_JOURNALMODE

TYPE	ENUM
DESCRIPTION	Sets the sqlite journal mode. Corresponds to the sqlite pragma journal_mode.
RANGE	“DELETE”, “TRUNCATE”, “PERSIST”, “MEMORY”, “WAL”, “OFF”
DEFAULT VALUE	“MEMORY”

CINEMO_OPTION_MM_SQLITE_INTEGRITY_CHECK

TYPE	ENUM
DESCRIPTION	Sqlite database integrity check to be performed on startup of Media Management. Corrupted databases will be removed automatically. Uses the sqlite pragma integrity_check. If set to “auto”, a full integrity check will be performed if CINEMO_OPTION_MM_SQLITE_JOURNALMODE is set to “OFF”, “MEMORY”, or “WAL”, otherwise no integrity check is performed.
RANGE	“auto”, “full”, “quick”, “off”
DEFAULT VALUE	“auto”

11.15 Apple Authentication Options

Most default values for Apple Authentication options are project specific and generic values cannot be provided. For further information please contact your Cinemo project leader.

CINEMO_OPTION_APPLEAUTH_URL

TYPE	STRING
DESCRIPTION	URL for communication with the Apple Authentication Coprocessor.
RANGE	Host system path syntax
DEFAULT VALUE	Project specific (e.g. "i2c:///dev/i2c-1:16")

CINEMO_OPTION_APPLEAUTH_LIBRARIES

TYPE	STRING
DESCRIPTION	Comma separated list of Apple Authentication library implementations.
RANGE	UTF-8
DEFAULT VALUE	Project specific

CINEMO_OPTION_APPLEAUTH_OPTION_STRING

TYPE	STRING
DESCRIPTION	Custom string, which is passed to each Apple Authentication library implementation upon creation.
RANGE	UTF-8
DEFAULT VALUE	Project specific

11.16 IAP Options

Most default values for iAP options are project specific and generic values cannot be provided. For further information please contact your Cinemo project leader.

CINEMO_OPTION_IAP_ENABLE_IAP1

TYPE	BOOLEAN
DESCRIPTION	Enable iAP 1 protocol.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_IAP_ENABLE_IAP2

TYPE	BOOLEAN
DESCRIPTION	Enable iAP 2 protocol.
RANGE	TRUE or FALSE
DEFAULT VALUE	TRUE

CINEMO_OPTION_IAP_POWER_FOR_DEVICE

TYPE	ENUM
DESCRIPTION	Specify how many power the hardware is capable to provide to the Apple device. This option is used only when the connection to the Apple device is wired (e.g. USB).
RANGE	OFF, 500 mA, 1000 mA, 1500 mA, 2100 mA, 2400 mA
DEFAULT VALUE	OFF

CINEMO_OPTION_IAP_RENDER_COVERART_AS_VIDEO

TYPE	BOOLEAN
DESCRIPTION	Render coverart from iAP device as video stream.
RANGE	TRUE or FALSE
DEFAULT VALUE	FALSE

CINEMO_OPTION_IAP_FFW_THRESHOLD

TYPE	INTEGER
DESCRIPTION	Speed threshold value for indication of trick play mode. This speed value will be reported in the CINEMO_EC_PLAYSPEED event. When the Apple device changes into fast forward mode by some user operation the value is positive. When the Apple device changes into fast backward mode by some user operation the negative amount of this value will be reported.
RANGE	1001 - 2147483647
DEFAULT VALUE	2500

CINEMO_OPTION_IAP_THREAD_SCHEDULE_DURATION

TYPE	INTEGER
DESCRIPTION	The (average) time in milliseconds it takes for a thread which is ready to run until it really runs on the system. This value is highly dependent on the characteristics of the BSP and is used to ensure tight timing requirements when acknowledgments of iAP-2 link data packets must be sent.
RANGE	0 - 1000
DEFAULT VALUE	10

CINEMO_OPTION_IAP_ACCESSORY_NAME

TYPE	STRING
DESCRIPTION	Fallback option for iAP authentication. Only used if not available from the metapool in the pconfig structure of the ICinemolAP::Open() call.
RANGE	UTF-8
DEFAULT VALUE	"Cinemo"

CINEMO_OPTION_IAP_ACCESSORY_MODEL_IDENTIFIER

TYPE	STRING
DESCRIPTION	Fallback option for iAP authentication. Only used if not available from the metapool in the pconfig structure of the ICinemolAP::Open() call.
RANGE	UTF-8
DEFAULT VALUE	"Cinemo iAP Accessory"

CINEMO_OPTION_IAP_ACCESSORY_MANUFACTURER

TYPE	STRING
DESCRIPTION	Fallback option for iAP authentication. Only used if not available from the metapool in the pconfig structure of the ICinemoIAP::Open() call.
RANGE	UTF-8
DEFAULT VALUE	“Cinemo GmbH”

CINEMO_OPTION_IAP_ACCESSORY_SERIAL_NO

TYPE	STRING
DESCRIPTION	Fallback option for iAP authentication. Only used if not available from the metapool in the pconfig structure of the ICinemoIAP::Open() call.
RANGE	UTF-8
DEFAULT VALUE	“SerialNo”

CINEMO_OPTION_IAP_ACCESSORY_FIRMWARE_VERSION

TYPE	STRING
DESCRIPTION	Fallback option for iAP authentication. Only used if not available from the metapool in the pconfig structure of the ICinemoIAP::Open() call.
RANGE	UTF-8
DEFAULT VALUE	“1.2.3”

CINEMO_OPTION_IAP_ACCESSORY_HARDWARE_VERSION

TYPE	STRING
DESCRIPTION	Fallback option for iAP authentication. Only used if not available from the metapool in the pconfig structure of the ICinemoIAP::Open() call.
RANGE	UTF-8
DEFAULT VALUE	“1.0.1”

CINEMO_OPTION_IAP_ACCESSORY_RF_CERTIFICATIONS

TYPE	INTEGER
DESCRIPTION	Value assigned by Apple during certification of the accessory. This option allows to specify the assigned value during accessory identification without the need to change the source code. If no such value has been assigned by Apple, then set it to zero (0).
RANGE	Value is a bit mask of 32 bits.
DEFAULT VALUE	0

CINEMO_OPTION_IAP_TRANSPORT_LIBRARIES

TYPE	STRING
DESCRIPTION	Comma separated list of transport library implementations.
RANGE	UTF-8
DEFAULT VALUE	Project specific

CINEMO_OPTION_IAP_TRANSPORT_OPTION_STRING

TYPE	STRING
DESCRIPTION	Custom string, which is passed to each transport library implementation upon creation.
RANGE	UTF-8
DEFAULT VALUE	Empty

CINEMO_OPTION_IAP_IAP1_RESTORE_ON_EXIT

TYPE	INTEGER
DESCRIPTION	Bit mask flagging the value for the 'restore on exit' parameter in the SetRepeat and SetShuffle iAP1 operations. Bit 1 is for SetRepeat, Bit 2 is for SetShuffle. If the bit is set (1) then the corresponding 'restore on exit' parameter has a non zero value of 0x01.
RANGE	0 - 3
DEFAULT VALUE	3

CINEMO_OPTION_IAP_IAP1_REPEAT_CYCLE_ORDER

TYPE	ENUM
DESCRIPTION	Specify the order in which ICinemoPlayer2::PostKeyUserUserEvent for CINEMO_KEY_REPEAT shall cycle through repeat states on iAP1 devices.
RANGE	off->all->single, off->single->all
DEFAULT VALUE	off->all->single

11.17 USB Options

CINEMO_OPTION_USB_ANDROIDAUTO_BLACKLIST

TYPE	BLOB
DESCRIPTION	AndroidAuto requires the support of AOAP. To detect whether a device is capable of AOAP, one "Get Protocol" request is send to the device. Some devices do not tolerate the request. To avoid breaking such devices, we define the option to prevent Cinemo from sending the request to the devices. The value of the option is a plain text with the following format: (<vendor-id>:<product-id><CR LF>)*, where <vendor-id> and <product-id> are hexadecimal numbers. If the <product-id> is 0, all the products of the vendor will be blocked.
DEFAULT VALUE	the default value is the internal blacklist maintained by Cinemo

CINEMO_OPTION_MTP_DEVICELIST

TYPE	BLOB
DESCRIPTION	The option is intended for customizing the behavior of Cinemo's MTP-implementation for special devices. For more details, please refer to the section MTP .
DEFAULT VALUE	the default value is the internal device-list tested by Cinemo well.

CINEMO_OPTION_MTP_UUID

TYPE	STRING
DESCRIPTION	The option is intended for generating the global unique persistent uuid for Samsung devices. For more details, please refer to the section MTP .
DEFAULT VALUE	"__seed__"

11.18 DLNA Options

This feature is supported by Cinemo but has not been enabled for your project Z101. If you are interested in using this feature, please contact Cinemo sales at sales@cinemo.com

HUMAX Confidential

11.19 Miscellaneous Options

CINEMO_OPTION_MISC_TS_PACKET_CALLBACK	
TYPE	CALLBACK
DESCRIPTION	<p>Specifies a callback function pointer which is used to provide the calling application with raw transport packets. The PIDs of the transport packets to be delivered to the application are specified as part of the parameter list parameter which is provided to CinemoCreateLiveStream().</p> <p>Note that due to the increased processing overhead it is not recommended to use this functionality for delivery of PIDs with high bitrate content.</p>
RANGE	Callback function pointer of type CinemoTSPacketCallback
DEFAULT VALUE	NULL

12. Cinemo Logging

Cinemo uses a powerful Logging Framework for debugging purposes and for tracing the execution state of the Cinemo Media Engine and the Cinemo Media Management Engine.

The Cinemo Logging Framework provides the following Features:

- Tree hierarchy-based log statements
- Multiple concurrently active outputs
- Large set of configurable output targets
- Fine-grained and dynamic changeable log level configuration
- Support for injecting user provided log messages
- Creation of compressed reports
- Size- and time-based file rolling
- Optional asynchronous delivery
- Optional XML-file-based log output configuration
- Support of GENIVI Diagnostic Log and Trace (DLT)

Throughout Cinemo's source code there are log statements providing information about the current state of execution. Each of these log statements is assigned to a path in the log tree and is emitted with a certain log level. In total there are seven log levels, ranging from *CINEMO_LOGLEVEL_VERBOSE* (very detailed information about the execution flow) up to *CINEMO_LOGLEVEL_FATAL* (severe problem happened). Based on the configured log levels for certain branches in the log tree, the application will only receive log messages of a particular severity and of a well defined subset of statements that are of interest.

By taking advantage of this fine-grained configurability, the amount of logging output can be significantly reduced while retaining the verbosity for such components which are of interest. All log statements which are disabled by the configuration (i.e. statements below the log level threshold or statements on a disabled part of the log tree) will not influence the code execution. Therefore, if the logging framework is configured properly, the system performance will not be affected.

Access to the Cinemo Logging Framework is provided by the following Interfaces:

- With [ICinemoLogAppender](#) an output target as well as multiple severity levels for respective sets of log statements and the log output layout can be specified. It is allowed to create and utilize multiple Log Appender instances at the same time with different targets and configuration.
- [ICinemoLogger](#) offers the possibility to inject own log statements into the Cinemo logging framework.

12.1 Targets for Log output

The following targets can be used by the ICinemoLogAppender instances:

- **File**

A File can be specified by means of a filename or by a file descriptor. Additional URL parameters for a file are:

- "compression=gzip" enables gzip compression
- "rollingMaxSizeBytes=<bytes>" enables file rolling (a new file is created, if the size of the current one exceeds the specified number of bytes)
- "rollingMaxDurationSeconds=<seconds>" enables file rolling (a new file is created, if the current one is older than <seconds>)
- "rollingMaxIndex=<imax>" specifies the maximum number of files until an old one is overridden
- "postfix=pid|tid|rand" adds the process id or thread id or a random number to the filename
- "async" enables asynchronous delivery of log messages

- **Pipe**

A Named Pipe can be used to efficiently deliver the log messages to a different process running at the system.

- **Standard Console Output**

C library stdout or stderr is used to output the log.

- **Callback**

User supplied Callback Function.

- **Syslog**

System-specific log output means (e.g. LogCat on Android OS).

- **Genivi DLT**

GENIVI Diagnostic Log and Trace.

- **Output Debug String**

Microsoft Windows Debugging Messages

12.2 XML File Based Configuration

The creation of log outputs can be enabled by supplying an XML configuration file. A default XML file `cinemo_log.xml` is supplied with the binaries provided as part of the Cinemo SDK. This file should be placed in the Cinemo plugin folder. By means of the configuration file, multiple output targets with a different log configuration can be enabled without the need of the programmatic utilization of the aforementioned functions of the Cinemo Media Engine API.

Please contact your Cinemo representative if you have questions about this feature.

13. Cinemo Sample Applications

The Cinemo SDK includes sample applications that are provided for the purpose of demonstrating how to use the Cinemo API. Although not intended for usage in a production environment, these applications provide a good starting point for developing individual solutions.

The following sections give a short overview of the the applications' usage and all options. In the last section, a short tutorial is provided to demonstrate possible use cases for the sample applications by example.

13.1 SampleAudioCodec

The application SampleAudioCodec decodes files from various formats and creates a wave file out of it, or encodes a wave file in AAC format.

13.1.1 Decoding Usage

```
SampleAudioCodec -d -i <src.(mp3|aac|ac3)> -o <out.wav>
```

Parameters

- **-d**
Set to decoding mode.
- **-i**
Name of the input file, encoded in MP3, AAC or AC3.
- **-o**
Name of the output file to write the decoded wave data to.

Remarks

For AAC encoded input data, only ADTS is supported.

13.1.2 Encoding Usage

```
SampleAudioCodec -e [-cbr] -b <bitrate/kbps> -i <src.wav> -o <out.(aac|m4a)>
```

Parameters

- **-e**
Set to encoding mode.
- **-cbr**
Encode using constant bitrate.
- **-b <bitrate/kbps>**
Set the bitrate of the encoded file.
- **-i**
Name of the input file in wave format.
- **-o**
Name of the output file to write the AAC-encoded audio data to.

Remarks

If the output filename has a '.mp4' extension, Cinemo will write the AAC frames inside an MPEG4 container. Otherwise, an ADTS file will be generated.

13.2 SampleBrowse

The application SampleBrowse recursively lists all child nodes within a UPnP tree.

Usage

```
SampleBrowse <uri>
```

Parameters

- <uri>

The URI provided with this option identifies a node in the UPnP tree of a UPnP server.

Remarks

The URI can for example have the following format:

"upnp://<server_ipaddress>:<server_port>/0/MediaServer/DeviceDesc.xml?pid=<node_id>&browse=BrowseDirectChildren"

The application will then list all child nodes recursively starting at the node provided by <node_id>.

13.3 SampleCopier

The application SampleCopier rips the audio tracks from an Audio-CD and encodes it using AAC.

Usage

```
SampleCopier <CD device> <target folder>
```

Parameters

- **<CD device>**

This is the CD device containing the Audio-CD to be ripped (for instance /dev/sdb).

- **<target folder>**

The encoded tracks will be stored to this folder.

13.4 SampleMetadata

The application SampleMetadata reads the metadata of the given URL and prints it on the console.

Usage

```
SampleMetadata <URL>
```

Parameters

- <URL>

The metadata of this URL or file is printed to stdout. This could for example be a filepath like "/mnt/media/song.mp3" or a UPnP URI like "upnp://<server_ipaddress>:<server_port>/0/MediaServer/DeviceDesc.xml?pid=<node_id>&browse=BrowseDirectChildren".

13.5 SampleMM

The application SampleMM instantiates a media server based on the Cinemo Media Management Engine.

Usage

```
SampleMM [<options>] <url> <...>
```

Parameters

- **<url>**

The URLs/folders that will be automatically mounted upon start of the media server.

- **-c <url>**

Connect to another instance that can be reached by the given URL.

- **-w [<url>]**

Watch for device insert and removal. You can restrict the watch to a specific folder by adding it as argument to this option.

- **-i <interface>**

Network interface to advertise UPnP services (for example, "eth0" on a Linux based system).

- **-u <url>**

Url to be used by the media server.

- **-n <name>**

Media server name used when announcing itself via SSDP

- **-d**

Use device flag when mounting volumes.

- **-h**

Display help.

- **-l <int>**

Set verbosity of the log output (0=trace, 1=debug, 2=info, 3=warn, 4=error, 5=fatal).

- **-s**

Redirect log message to the standard output instead of the error output.

- **-q**

Quiet mode: No prompt will appear and the application will run in standalone mode.

After starting SampleMM, a command prompt will appear that enables a user to control the media server by entering following commands (Please note that no prompt will appear if you use the '-q' option).

Commands

- **q | quit | exit**

Shut down and exit the media server.

- **sync <folder>**

Mount a volume based on the given folder and perform synchronization (indexing, meta data extraction and thumbnail-generation).

- **remove <int>**

Remove the volume identified by the given node id <int>. The node id can for example be determined by using SampleMetadata to browse the existing volumes of the media server.

- **title <int> <name>**

Change the VFS_NAME of the volume provided by the node id <int> to the string provided by <name>.

- **reindex**

Reindex all mass storage device volumes.

13.6 SampleMMPhonetic

The application SampleMMPhonetic demonstrates phonetic queries based on the Cinemo Media Management Engine.

Usage

```
SampleMMPhonetic [<options>] <url> <...>
```

Parameters

- **<url>**
The URLs/folders that will be automatically mounted upon start of the media server.
- **-u <url>**
Url to be used by the media server.
- **-n <name>**
Media server name used when announcing itself via SSDP.
- **-h**
Display help.
- **-l <int>**
Set verbosity of the log output (0=trace, 1=debug, 2=info, 3=warn, 4=error, 5=fatal).
- **-c**
Colored output of text matches.
- **-v**
Verbose mode.

If no <url> is specified, a sample data set will be used.

After starting SampleMMPhonetic, a command prompt will appear that enables a user to perform phonetic pinyin queries on the media server by entering the following commands.

Commands

- **q | quit | exit**
Shut down and exit the media server.
- **phonetic <string>**
Search tracks by title using phonetic matching.
- **any <string>**
Search tracks by title using combined phonetic and written matching.
- **mixed <written>,<any>**
Search tracks by mixed written and combined phonetic matching. <written> will match in written field, <any> will match in phonetic or written field of subsequent characters.

13.7 SampleMMRemoteIndexing

The application SampleMMRemoteIndexing demonstrates remote indexing of a Cinemo Media Management server.

Usage

```
SampleMMRemoteIndexing [<options>] <url>
```

Parameters

- **<url>**
The url of a remote server to mount on the local server.
- **-h**
Display help.
- **-l <int>**
Set verbosity of the log output (0=trace, 1=debug, 2=info, 3=warn, 4=error, 5=fatal).

The following prerequisites need to be met on the remote server side:

- **Enable journaling**
Indexing of remote servers requires journaling to be enabled on the remote server by means of the option *CINEMO_OPTION_MM_ENABLE_JOURNALING*.
- **Assign server uuids**
Each MM server needs to be assigned a unique and persistent uuid by means of the option *CINEMO_OPTION_MM_SERVER_UUID*.

The *<url>* parameter points to the remote server and must include the *cinemo_udn* parameter, e.g:

```
SampleMMRemoteIndexing "upnp://127.0.0.1:40000/0/MediaServer/DeviceDesc.xml?cinemo_udn=uuid:<uuid>"
```

where *<uuid>* corresponds to the *CINEMO_OPTION_MM_SERVER_UUID* of the remote server.

13.8 SamplePlayer

The application SamplePlayer plays one or multiple files provided as URLs.

Usage

```
SamplePlayer [<options>] <URL> <...>
```

Parameters

- <**URL**> <...>

The URL(s) of the media to be played.

13.9 SamplePlaylistSelect

SamplePlaylistSelect demonstrates the use of the ICinemoPlaylist::Select() command. It can be used to send the contents of an ICinemoPlaylist to the underlying playback device. Currently this is implemented for iAP and AVRCP.

The ICinemoPlaylist::Select() command can be used in two different scenarios. Both of them will be described briefly in the following paragraphs.

Scenario 1 – Local device

This is the case, where a local device is browsed via a VFS, e.g. an iAP1 device. A typical scenario could be the following:

As soon as an iAP1 device is connected, an ICinemoPlaylist is opened with the device URL and playback is started with the first track, the “Now Playing” track. To monitor the playback list of the device, a second playlist can be opened for the “Now Playing” playlist.

For browsing the media library on the device, a third ICinemoPlaylist can be used. If a request to play the current items in this third playlist is made, the ICinemoPlaylist::Select() command can be used. It will pass the current browsing context of the third playlist to the underlying iAP device and request playback. As a result, the “Now Playing” track in the ICinemoPlayer as well as the contents in the second playlist should be updated.

Scenario 2 – Browsing via UPnP

The second scenario describes the case where local browsing is not available, e.g. on iAP2 devices. Their media library is only available in a special format, which cannot be browsed like a regular filesystem without additional effort. Usually Cinemo Media Management is used to browse iAP2 devices.

The use case is very similar to the first scenario. Except for browsing the media library directly on the device, a UPnP server is used. While this also enable additional features like searching and sorting, a playlist generated from a UPnP query can be selected in the same way as a playlist generated by browsing.

Usage

```
SamplePlaylistSelect [<options>] <url>
```

Parameters

- **-h**
Show help message.
- **-l <int>**
Set log level (0..3).
- **-s**
Set log output to stdout.
- **-t**
Use SelectTrack instead of Select.

- <url>

URL to open the playlist with.

Remarks

The URL is mandatory and can be any of the following:

- a device path
- a UPnP browsing URL
- a UPnP search query

The argument of the –t flag is the one-base track index in the playlist.

13.9.1 Example for local browsing

This example will start playback of the “Now Playing” track of an iAP 1 device and request the playback of the second song of the first album on the device with the ICinemoPlaylist::SelectTrack() call.

For this example to work correctly, the iAP2 support of Cinemo needs to be disabled in `cinemo_options.xml`:

```
<iAP>
  <EnableIAP2>0</EnableIAP2>
</iAP>
```

Step 1

Determine the USB path of the Apple device using `SampleMetadata`:

```
# ./SampleMetadata usb://*
Cinemo Media Engine v1.60.0.0

opening...
path: usb://*
type: FOLDER | VIRTUAL | METADATA_COMPLETE | WATCHABLE
title 01 index 00 VFS_PATH: "iap://usb:///dev/bus/usb/001/006"
title 01 index 00 VFS_NAME: "iPod"
title 01 index 00 VFS_TYPE: FOLDER | DEVICE | VIRTUAL | ORDERED | METADATA_COMPLETE |
WATCHABLE | USB | IAP
title 01 index 00 VFS_USB_ID: 512
title 01 index 00 VFS_USB_DEVICE_CLASS: 0
title 01 index 00 VFS_USB_DEVICE_SUBCLASS: 0
title 01 index 00 VFS_USB_DEVICE_PROTOCOL: 0
title 01 index 00 VFS_USB_MAX_PACKET_SIZE: 64
title 01 index 00 VFS_USB_VENDOR_ID: 1452
title 01 index 00 VFS_USB_PRODUCT_ID: 4778
title 01 index 00 VFS_USB_DEVICE_ID: 1296
title 01 index 00 VFS_USB_MANUFACTURER: "Apple Inc."
title 01 index 00 VFS_USB_PRODUCT: "iPod"
title 01 index 00 VFS_USB_SERIAL_NO: "9323f85c9dbb402a1b675a4845626f89094518bd"
title 01 index 00 VFS_USB_NUM_CONFIGURATIONS: 4
title 00 index 00 VFS_INDEX: 1
title 00 index 00 VFS_COUNT: 1
title 00 index 00 VFS_TOTAL: 1

watching...
```

In this case it is “`iap://usb:///dev/bus/usb/001/006`”.

Step 2

Start a DDP server on the IPv6 localhost address:

```
# ./SampleDDP ddp://[::1]:30001
```

Step 3

Start the playback of the “Now Playing” track using SamplePlayer.

```
# ./SamplePlayer "ddpiaptrack://[::1]:30001?PATH=iaptrack%3A%2F%2Fusb%3A%2F%2F%2Fdev%2Fbus%2Fusb%2F001%2F006"
```

The URL is created by appending the url-encoded path of the “Now Playing” track (iaptrack://usb:///dev/bus/usb/001/006) to the DDP base url as the PATH parameter.

Step 4

Select the first album on the device.

```
# ./SamplePlaylistSelect -t 2 "ddp://[::1]:30001?PATH=iap%3A%2F%2Fusb%3A%2F%2F%2Fdev%2Fbus%2Fusb%2F001%2F006%3Fpid%3D1%26ctx%3D1-3-0-5"
```

The path for the first album of the device on an iAP1 device is iap://usb:///dev/bus/usb/001/006?pid=1&ctx=1-3-0-5. This is added as a url-encoded parameter to the DDP base url.

The parameter -t 2 requests the playback to start at the second item in the playlist, which corresponds to the second song of the album.

Step 5

Verify the first album is playing by checking the SamplePlayer or the display of the Apple device.

13.9.2 Example for remote browsing

This example will start playback of the “Now Playing” track of an iAP 2 device and request the playback of all songs which match the search string “madonna” with the ICinemoPlaylist::Select() call.

For this example to work correctly, the following options need to be set in `cinemo_options.xml`:

```
<iAP>
  <EnableIAP2>1</EnableIAP2>
</iAP>
<MM>
  <Server>
    <Port>30002</Port>
  </Server>
  <DDP>
    <ServerUrl>ddp://[::1]:30001</ServerUrl>
  </DDP>
</MM>
```

The MM server will be running on port 30002 and the DDP server (inside the MM server) on port 30001.

Step 1

```
# ./SampleMetadata usb://*
Cinemo Media Engine v1.60.0.0

opening...
path: usb://*
type: FOLDER | VIRTUAL | METADATA_COMPLETE | WATCHABLE
title 01 index 00 VFS_PATH: "iap://usb:///dev/bus/usb/001/007"
title 01 index 00 VFS_NAME: "iPod"
title 01 index 00 VFS_TYPE: FOLDER | DEVICE | VIRTUAL | ORDERED | METADATA_COMPLETE |
WATCHABLE | USB | IAP
title 01 index 00 VFS_USB_ID: 512
title 01 index 00 VFS_USB_DEVICE_CLASS: 0
title 01 index 00 VFS_USB_DEVICE_SUBCLASS: 0
title 01 index 00 VFS_USB_DEVICE_PROTOCOL: 0
title 01 index 00 VFS_USB_MAX_PACKET_SIZE: 64
title 01 index 00 VFS_USB_VENDOR_ID: 1452
title 01 index 00 VFS_USB_PRODUCT_ID: 4778
title 01 index 00 VFS_USB_DEVICE_ID: 1296
title 01 index 00 VFS_USB_MANUFACTURER: "Apple Inc."
title 01 index 00 VFS_USB_PRODUCT: "iPod"
title 01 index 00 VFS_USB_SERIAL_NO: "9323f85c9dbb402a1b675a4845626f89094518bd"
title 01 index 00 VFS_USB_NUM_CONFIGURATIONS: 4
title 00 index 00 VFS_INDEX: 1
title 00 index 00 VFS_COUNT: 1
title 00 index 00 VFS_TOTAL: 1

watching...
```

In this case it is “`iap://usb:///dev/bus/usb/001/007`”.

Step 2

Start the MM server and add the Apple device:

```
# ./SampleMM iap://usb:///dev/bus/usb/001/007
```

Step 3

Start the playback of the “Now Playing” track.

```
# ./SamplePlayer "ddpiaptrack://[::1]:30001?PATH=iaptrack%3A%2F%2Fusb%3A%2F%2F%2Fdev%2Fbus%2Fusb%2F001%2F007"
```

The URL is created by appending the url-encoded path of the “Now Playing” track (iaptrack://usb:///dev/bus/usb/001/007) to the DDP base url as the PATH parameter.

Step 4

Select the first album on the device:

```
# ./SamplePlaylistSelect
"upnp://[::1]:30002/0/MediaServer/DeviceDesc.xml?pid=0& \
search=(upnp:class derivedfrom \"object.item\" and (\ \
dc:title contains \"madonna\" or \
upnp:album contains \"madonna\" or \
upnp:artist contains \"madonna\" or \
upnp:albumArtist contains \"madonna\") and (@refID exists false))"
```

The UPnP query searches for all items on the MM server containing the string “madonna”.

Step 5

Verify all songs from madonna are in the “Now Playing” playlist by checking the display of the Apple device.

13.10 SampleDDP

The SampleDDP application starts a Cinemo DDP server.

Usage

```
SampleDDP <URL>
```

Parameters

- <URL>

The URL of the Cinemo DDP server to be started.

Example: To start a Cinemo DDP server with the IP address 192.168.1.156 and the port number 1234, call

```
SampleDDP ddp://192.168.1.156:1234
```

It is possible to start the server on all available network interfaces with the following command:

```
SampleDDP ddp://@:1234
```

13.11 SampleWindowBitmap

The application SampleWindowBitmap demonstrates how to use ICinemoWindowBitmap in order to resize images. By default, it writes an output image out.jpg with size 100x100.

Usage

```
SampleWindowBitmap [<options>] <URL>
```

Parameters

- <URL>

The URL of the image to be resized.

14. Cinemo Tutorials

The following tutorials are intended to provide guidelines for implementing common use cases with the Cinemo Media Engine.

SECTION	DESCRIPTION
Android Auto	Tutorials for advanced use cases related to Android Auto
Apple Devices	Tutorials for advanced use cases involving Apple devices
DDP	Accessing remote devices using the DDP protocol
Distributed Playback	How do discover Distributed Playback sessions
Media Server	How to start, browse and play content from Cinemo Media Management

14.1 AndroidAuto

14.1.1 Connecting a VEC session remotely

Android Auto provides support for so-called Vendor Extension Channels (VEC), which Google describes as follows:

"A vendor extension endpoint is one that utilizes GAL only as a transport. Vendor extension channels are completely opaque to GAL and the bits are not interpreted, they are simply sent over the wire as is. This allows for OEMs to write proprietary services that can run over GAL alongside with the regular GAL service endpoints while enjoying all the benefits that GAL provides."

Cinemo fully supports the VEC feature and allows access to individual channels from different processes.

Getting the VEC path

After an Android Auto session has been established, the metapool of the corresponding `ICinemoPlayer` instance will contain information about available channels:

```
===== Player Metapool =====
[ 0|00]      VFS_BIND_PATH: "/dev/bus/usb/003/012"
[ 0|00] AndroidAuto_VEC_Path: "aapvec://usb:///dev/bus/usb/003/012?id=0"
[ 0|00] AndroidAuto_VEC_Service_Name: "com.vendorname.servicename"
[ 1|00] AndroidAuto_VEC_Path: "aapvec://usb:///dev/bus/usb/003/012?id=1"
[ 1|00] AndroidAuto_VEC_Service_Name: "com.vendorname.servicename2"
=====
```

The `CINEMO_METANAME_ANDROIDAUTO_VEC_PATH` metaname contains the actual path of the VEC and `CINEMO_METANAME_ANDROIDAUTO_VEC_SERVICE_NAME` the channel name. Multiple channels will be reported as different titles.

Opening the ICinemoDataPort

Each channel may be opened by one `ICinemoDataPort`. After the interface has been created, call `ICinemoDataPort::Open()` to connect the `ICinemoDataPort`.

Possible error codes for `ICinemoDataPort::Open()` are:

- **CinemoErrorNotConnected**

No active Android Auto session found.

- **CinemoErrorState**

Another `ICinemoDataPort` has already claimed this channel.

- **CinemoErrorConfiguration**

An invalid channel index has been requested.

Communication with the app on the Android device can now be achieved using the `ICinemoDataPort` methods.

Opening the ICinemoDataPort remotely

It is possible to connect to a VEC session from a different process using DDP. The process handling the [ICinemoAndroidAuto](#) instance needs to run a DDP server for this to work.

The URL necessary for opening the remote [ICinemoDataPort](#) can be created from the CINEMO_METANAME_ANDROIDAUTO_VEC_PATH metapool entry and the DDP server address. The correct protocol for VEC via DDP is “ddpdataport”.

Assuming the DDP server is running on 192.168.1.1:1234, the URL would become

```
ddpdataport://192.168.1.1:1234?PATH=aapvec%3A%2F%2Fusb%3A%2F%2F%2Fdev%2Fbus%2Fusb%2F003%2F012?  
id=0
```

14.2 Apple Devices

Apple devices provide great user experiences when used directly. Cinemo tries to stick very closely to their behavior to keep up this experience. But due to the large number of different Apple devices available, different approaches for a single problem may be required to maintain the usability of the device through Cinemo. In this section common requests related to iAP devices and the recommended solution will be presented.

14.2.1 Searching iAP1 devices

Using the iAP1 protocol as provided by Apple, it is not possible to perform any search queries on the devices' media library. Apple only provides a static, browseable device hierarchy for such devices.

Cinemo Media Management allows indexing the media library of most iAP1 devices. Using UPnP queries, it is then possible to search the database with custom search expressions.

However, there are iAP1 devices, which are not supported by Cinemo Media Management. This is due to missing commands, which are required for efficiently indexing the device. For such devices only the device hierarchy provided by Apple is available, which cannot be searched. In order to provide the search functionality, the following approach is recommended by Cinemo.

Detecting device support

It is important to determine the capabilities of an Apple device right after it has been connected. There are three different levels of browse support:

1. device can be indexed by Cinemo Media Management
2. device cannot be indexed but does provide the database engine
3. device cannot be indexed and does not provide the database engine

The level of a device can be identified using the device metapool. It may look like the following:

```
path: iap://usb:///dev/bus/usb/003/006
type: FOLDER | VIRTUAL | ORDERED | WATCHABLE | SELECTABLE | USB | IAP | iAP-1
title 00 index 00 VFS_USB_VENDOR_ID: 1452
title 00 index 00 VFS_USB_PRODUCT_ID: 4706
title 00 index 00 VFS_USB_MANUFACTURER: "Apple Inc."
title 00 index 00 VFS_USB_PRODUCT: "iPod"
title 00 index 00 VFS_USB_SERIAL_NO: "000A27001B060F39"
title 00 index 00 VFS_UUID: "000A27001B060F39_iAP1"
title 00 index 00 VFS_NAME: "Cinemo's iPod nano 3rd (HW-528)"
title 00 index 00 DeviceIdentificationInfo: "YM746U2GYXV"
title 00 index 00 VFS_IAP1_IPOD_VERSION: "1.1.3"
title 00 index 00 VFS_IAP1_LINGO_VERSION: 263
title 00 index 03 VFS_IAP1_LINGO_VERSION: 261
title 00 index 04 VFS_IAP1_LINGO_VERSION: 269
title 00 index 10 VFS_IAP1_LINGO_VERSION: 259
title 00 index 00 VFS_MAY_BE_INDEXED: 0
title 00 index 00 VFS_IAP_FLAGS: (40000000h) BROWSING_HIERARCHY_AVAILABLE
title 01 index 00 VFS_PATH: "iaptrack://usb:///dev/bus/usb/003/006"
title 01 index 00 VFS_NAME: "NowPlaying"
```

```

title 01 index 00 VFS_TYPE: FILE | VIRTUAL | PLAYABLE | SELECTABLE | USB | IAP | iAP-1
title 01 index 00 VFS_NOWPLAYING_TYPE: 2
title 02 index 00 VFS_PATH: "iap://usb:///dev/bus/usb/003/006?pid=0"
title 02 index 00 VFS_NAME: "NowPlaying"
title 02 index 00 VFS_TYPE: FOLDER | VIRTUAL | ORDERED | WATCHABLE | SELECTABLE | USB | IAP |
    iAP-1
title 02 index 00 VFS_NOWPLAYING_TYPE: 1
title 03 index 00 VFS_PATH: "iap://usb:///dev/bus/usb/003/006?pid=1"
title 03 index 00 VFS_NAME: "Cinemo's iPod nano 3rd (HW-528)"
title 03 index 00 VFS_TYPE: FOLDER | VIRTUAL | ORDERED | WATCHABLE | SELECTABLE | USB | IAP |
    iAP-1
title 00 index 00 VFS_INDEX: 1
title 00 index 00 VFS_COUNT: 3
title 00 index 00 VFS_TOTAL: 3

```

If the device may be indexed using Cinemo Media Management, the `VFS_MAY_BE_INDEXED` entry will be set to 1. If this entry is set to 0, the database engine support needs to be checked. All devices which support the Extended Interface Lingo (0x04) Version 1.13 or higher have database engine support. The version for the current device is encoded in the `VFS_IAP1_LINGO_VERSION` with the index 04. A value of 269 or higher corresponds to at least version 1.13. Devices which only support a lower version can neither be indexed nor provide the database engine.

For some device the `VFS_IAP1_LINGO_VERSION` entry is not available. Instead, the `VFS_IAP1_LINGO_OPTIONS` may be used to determine the capabilities. Like for the `VFS_IAP1_LINGO_VERSION`, the item with index 04 represents the "Extended Lingo". If the second bit of this bitmask is set (0x2), the device supports the database engine.

Devices supporting indexing

For devices which can be indexed using Cinemo Media Management there is nothing left to be done. Search queries can be executed using UPnP query expressions as described in [Query Language](#).

Playlists which have been opened with a UPnP query may be transferred to the device using the [ICinemoPlaylist::Select\(\)](#) or [ICinemoPlaylist::SelectTrack\(\)](#) methods.

Devices supporting the database engine

For devices which cannot be indexed but do support the database engine it is required to store metadata for all tracks in a local database, which may then be used for evaluating search queries. Cinemo does not implement this approach, because of its drawbacks, which are described in [this](#) paragraph.

The database engine of an Apple device allows the extraction of metadata for all tracks independently from the current playing track. This should be done using an [ICinemoPlaylist](#) which has been opened with the "Tracks" folder on the device. For iAP1 this is always the following path:

```
iap://<url of the device>?pid=1&ctx=1-5
```

The playlist will contain information about all tracks on the device, which can be retrieved using the [ICinemoPlaylist::GetMetapool\(\)](#) method. Internally this playlist uses paging when requesting the data in order to minimize the workload on the Apple device.

Please note that the playlist will only retrieve the name and the path of each track but not the entire metadata. This allows displaying the list very quickly without having to wait for the metadata for each track.

For retrieving all available metadata for each track, an [ICinemoVFS](#) may be used. Calling [ICinemoVFS::Open\(\)](#) with the path from the playlist metapool will open a VFS for that particular track which contains the entire metadata which is available for that track.

When artwork is not needed, the process can be sped up by adding the URL parameter “artwork=0”. This disables fetching of coverart, which usually takes up most of the time.

From the metadata gathered from all tracks a custom track database can be built. Search requests should then be handled by this database.

Starting playback from custom search requests, however, is very difficult and only works with some [restrictions](#).

Devices without database engine support

For devices which cannot be indexed and also do not support the database engine extracting the track metadata is possible with some restrictions. Because only metadata from tracks in the “Now Playing” playlist can be retrieved, loading all tracks into the “Now Playing” playlist is required. This means that effectively the device needs to be blocked until the operation has finished and the user cannot interact with the device during this time. Usually this process takes few minutes but for devices containing lots of tracks it make take up to ten minutes to finish.

Before starting to retrieve the metadata, all tracks need to be loaded into the “Now Playing” playlist. This can be achieved by calling [ICinemoPlaylist::Select\(\)](#) on a playlist opened with the “Tracks” folder.

Similarly like for devices supporting the database engine, the metadata for all tracks may then be retrieved from the “Now Playing” playlist, available at

```
iap://<url of the device>?pid=0
```

From the metadata gathered from all tracks a custom track database can be built. Search requests should then be handled by this database.

Starting playback from custom search requests, however, is very difficult and only works with some [restrictions](#).

Restrictions for custom playlists

Sending custom search requests to an iAP1 device is only possible using track UIDs, which are only available if the device can be indexed using Cinemo Media Management. For the remaining two cases only playlists/containers which already exist on the device can be used for playback commands.

Tracks as search results If the search result consists of individual tracks, the list cannot be loaded onto the Apple device.

Containers as search results If the search result consists of containers, like e.g. an album or an artist, this may be sent to the Apple device as follows. For this example we will assume the search query returned a single album.

In order to start playback of this album, identify the index of this album in the “Albums” container by comparing the name. The “Albums” container is always available at this path:

```
iap://<url of the device>?pid=0&ctx=1-3
```

Open an [ICinemoPlaylist](#) for this container and call [ICinemoPlaylist::SelectTrack\(\)](#) with the index of the album as the argument.

Unfortunately this only works if the search query returns a single item. If two or more items are returned this approach does not work anymore.

14.2.2 Highlighting the currently playing track

A commonly requested feature, which can be implemented in many different ways is the highlighting of the currently playing track in the browse hierarchy. Please note that it is not possible to perform this highlighting for playlists which contain the “NowPlaying” track due to limitations of the iAP protocol.

While some approaches work rather well and quick, there are some which would create lots of unnecessary communication with the device itself. The following solutions are the recommended way to implement such a feature.

Browsing based on Cinemo Media Management

If device browsing is realized using Cinemo Media Management, highlighting basically comes for free. There are a few simple steps which will provide all information necessary.

Getting the UID of the “NowPlaying” track If the Apple device does provide a UID for the “NowPlaying” track it will be available in the metapool of the attached Cinemo Player. The corresponding metaname is **CINEMO_METANAME_VFS_IAP_UID**. The UID uniquely identifies the track on the Apple device and in the Cinemo Media Management.

Search for the UID With the UID from the metapool of the “NowPlaying” track, the node ID in the Cinemo Media Management can be retrieved. Using a UPnP query on the volume node of the indexed iAP volume should yield the node ID:

```
upnp://<ip>:<port>/0/MediaServer/DeviceDesc.xml?pid=<volume node ID>&search=(cinemo:iapUID=<iAP UID>)
```

The result of this query will contain the **CINEMO_METANAME_VFS_UPNP_ID**, which is the input parameter for the [ICinemoMM::GetHierarchyNodes\(\)](#) method. The returned [ICinemoMetapool](#) will contain all hierarchy

containers, this node is linked in. If any of these containers is currently being displayed on the screen, it may be highlighted.

Browsing based on the device browse hierarchy

When using the device browse hierarchy, the track UID cannot be used for highlighting for several reasons. First, not all devices which can be browsed do provide track UIDs. Especially older Apple devices do not support track UIDs. Another point is that in higher levels of the hierarchy like “Album” or “Artist” the UID is not available. While iAP2 theoretically provides Collection IDs for Artist, Albums, etc., these are not provided in the playback engine for the “NowPlaying” track. The most efficient approach for solving this issue is using simple string comparisons with the “NowPlaying” track.

Getting the “NowPlaying” track metadata First all metadata of the “NowPlaying” track needs to be extracted from the metapool. The set of required data should match all hierarchy containers, which usually is “Album”, “Artist”, “Composer” and “Genre”.

Highlighting a matching container In order to efficiently highlight a container, it is not advisable to search for the “NowPlaying” track in the hierarchy. Instead each item which is being displayed should be compared if it matches the criteria extracted from the “NowPlaying” track and then be highlighted.

Here is a short example of how this could be implemented:

- Get the metadata from the “NowPlaying” track, e.g. “Artist” = “Madonna”
- If the user browses “All Artists”, get the list of all artists
- Determine which items of the list are currently being display on the HMI
- Check for each of the displayed items if the “Artist” equals “Madonna”
- Apply the highlighting to the matching item

14.2.3 Remote access for the Apple Authentication chip

Cinemo provides an easy way for thread-safe access of an Apple Authentication chip from different threads, processes or even hosts with the use of the DDP protocol.

Setting up the server

The process which shall act as the authentication server needs to start the DDP server. Additionally, the CINEMO_OPTION_APPLEAUTH_URL needs to be set to the actual chip, e.g.

- `ICinemoConfig::SetOption(CINEMO_OPTION_APPLEAUTH_URL, "i2c://dev/i2c-1:16")`

Setting up the clients

In each client the authentication URL needs to be set to the plain DDP server URL like this:

- `ICinemoConfig::SetOption(CINEMO_OPTION_APPLEAUTH_URL, "ddp://127.0.0.1:1234")`

All authentication requests from clients will be forwarded to the authentication chip configured in the DDP server process and not processed locally. This guarantees exclusive access to the chip.

14.3 DDP

DDP is Cinemo's Distributed Device Protocol. This is a protocol which is similar to HTTP. It allows a (Cinemo) DDP client to communicate with a (Cinemo) DDP server over a network.

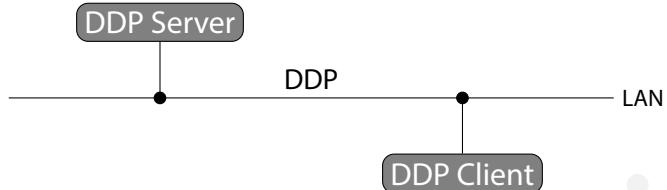


Figure 14.1: Overview DDP

Example: the (Cinemo) DDP server is a device which contains a Blu-ray drive with a Blu-ray disc, the (Cinemo) DDP client is a PC without Blu-ray drive. The DDP client can play the Blu-ray disc of the DDP server by sending drive commands over DDP to the DDP server; the DDP server executes the drive commands and sends their result back to the DDP client; the DDP client plays the content of the Blu-ray disc received from the DDP server.

Other scenarios are that the DDP server contains a CD/DVD drive with a disc or a hard disk with audio files.

14.3.1 DDP Server

Assume that your (Cinemo) DDP server has the IP address 192.168.1.156 and the port number 1234. Call the function *CinemoCreateDDP()* defined in *cinemo.h* to create an object of type [ICinemoDDP](#), then call *CreateServer("ddp://192.168.1.156:1234")* to create the DDP server.

Example: a DDP server is created and started by calling the sample application SampleDDP with the DDP protocol and with the IP address and the port number of the DDP server as argument:

```
SampleDDP ddp://192.168.1.156:1234
```

To use all network interfaces, call

```
SampleDDP ddp://@:1234
```

14.3.2 DDP Client

A (Cinemo) DDP client is created and started like a usual Cinemo player with the following exception: the *Open* method or the *AppendTrack* method of [ICinemoPlaylist](#) extend their argument by the DDP protocol and the IP address and the port number of the DDP server the client wants to connect to.

Example: Assume that your (Cinemo) DDP server has the IP address 192.168.1.156 and the port number 1234. On the DDP server, there is an MP3 file "c:/Music/Mozart.mp3".

Suppose that you want to play the file "c:/Music/Mozart.mp3" on the DDP server with the SamplePlayer which is installed on a different device. Then create a DDP client by calling:

```
SamplePlayer ddpfile://192.168.1.156:1234?PATH=c%3A%2FMusic%2FMozart.mp3
```

Please note that the actual path needs to be URL-encoded.

Next, assume that the DDP server has a DVD drive "d:/" which contains a DVD. Then you can play the DVD of the DDP server by calling:

```
SamplePlayer ddpdisc://192.168.1.156:1234?PATH=d%3A%2F
```

14.3.3 Apple Devices

It is possible to access one or some Apple devices from a DDP client if the Apple devices are connected to a DDP server:

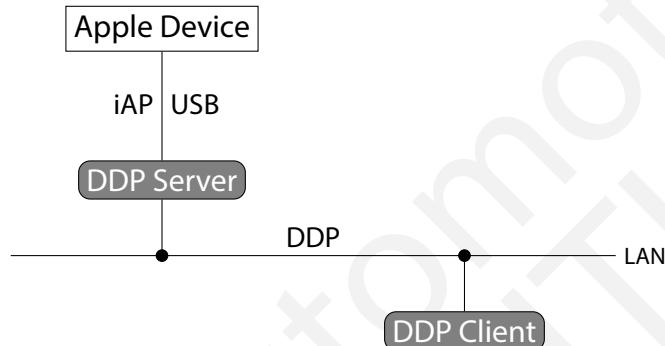


Figure 14.2: Overview DDP with a Apple device

In the following, assume that the DDP server has the IP address 192.168.1.156 and the port number 1234.

To search for all USB devices, the DDP client calls:

- `ICinemoVFS::Open("ddp://192.168.1.156:1234?PATH=usb%3A%2F%2F*")`

Note that you have to use URL-Encoding.

To open the playlist of a specific Apple device (namely "000/001"), the DDP client calls

- `ICinemoPlaylist::Open("ddp://192.168.1.156:1234?PATH=iap%3A%2F%2Fusb%3A%2F%2F000%2F001")`

Again, you have to use URL-Encoding in the above call.

Example: To create the DDP server, call

```
SampleDDP ddp://192.168.1.156:1234
```

When using SampleIAP or SampleIAPPPlayer as DDP client, call

```
SampleIAP -ac -r 192.168.1.156:1234
```

or

```
SampleIAPPPlayer -a -r 192.168.1.156:1234
```

14.3.4 DDP Security

By default, DDP transports its data plain and unauthenticated over the network. However, if the customer wants it, the whole communication over DDP can be encrypted and authenticated. Contact Cinemo to supply a version with DDP Security enabled (then Cinemo will release a version with enabled compiler switch ENABLE_DDP_SECURITY).

The following cryptographic primitives are used for DDP Security:

- Ed25519 for signature generation and verification
- Curve25519 for Diffie-Hellman key exchange
- ChaCha GCM (Galois Counter Mode) for data authentication and data encryption

It is possible to replace ChaCha by AES or Salsa20.

For each DDP connection, a separate authentication is executed. If the number of DDP connections is big, the performance of DDP Security is therefore slow. A bad performance in such a case can be improved considerably if authentications will be reused by session caching. Contact Cinemo if you want a version with Session Caching enabled (then Cinemo will release a version with enabled compiler switch ENABLE_DDP_SECURITY_CACHE).

14.4 Distributed Playback

The following sections describe how to use Cinemo Distributed Playback. All [ICinemoPlayer](#) instances, if configured to do so, will advertise themselves on the network as Distributed Playback servers, in the same way as [ICinemoMM](#) instances advertise themselves as Media Servers.

14.4.1 Discovery

If the IP address and port number of a Distributed Playback server is already known (i.e. because the player was configured with this information), then it is not necessary to perform discovery. Otherwise, in the same way as for media servers, it is possible to discover a distributed playback server using the "SampleMetadata" application, by entering the following command line:

```
SampleMetadata "ssdp://239.255.255.250:1900?urn=urn:schemas-upnp-org:device:CinemoMaster:1"
```

14.4.2 Playback

After discovering the IP address and port of the distributed playback master, it is possible to join the playback using the "SamplePlayer" application by entering the following command line:

```
SamplePlayer "slave://tcp://<serverip>:<serverport>"
```

The <serverip> and <serverport> parameters are obtained from the metadata displayed by the previous *SampleMetadata* command.

14.5 MediaServer

The following sections describe how to use Cinemo Media Management, which is implemented as a standalone UPnP media server with Cinemo extensions.

14.5.1 Starting a Media Server

We need to start a media server. Using the Cinemo sample applications, a media server is instantiated using the “SampleMM” application:

```
SampleMM -u @:40000 <media folder>
```

As written in the previous section describing the “SampleMM” application with a little bit more details, once the server is successfully created, you should see a command prompt.

Remarks

For the rest of this tutorial, we will assume that the machine we started the server on has an IP address of 192.168.0.1.

14.5.2 Discovering a Media Server

To discover a media server inside your network, you can use the “SampleMetadata” application. Using the SSDP protocol to trigger a broadcast that causes all media servers to announce themselves, this sample application will show the metadata associated to every media server on the console

This broadcast can be triggered by entering:

```
SampleMetadata "ssdp://239.255.255.250:1900?urn=urn:schemas-upnp-org:device:MediaServer:1"
```

Remarks

In the previous section, we have fixed the port of the media server using the ‘-u’ option when starting the “SampleMM” application. When you don’t fix a port or when you don’t know the IP address of the media server in advance, using the above command line will show it to you on the console.

14.5.3 Browsing a Media Server

Once the media server is running, its contents can be browsed using the “SampleBrowse” application with the following command line:

```
SampleBrowse "upnp://192.168.0.1:40000/0/MediaServer/DeviceDesc.xml?pid=0&browse=BrowseDirectChildren"
```

Remarks

By setting the parameter “pid” to zero, browsing will start recursively at the root node.

14.5.4 Playing Media from a Media Server

In order to play media stored on the server, you will first need to get its URL. This can be achieved once again using the “SampleMetadata” application, by entering the following command line:

```
SampleMetadata "upnp://192.168.0.1:40000/0/MediaServer/DeviceDesc.xml?pid=<media_pid>&browse=BrowseDirectChildren"
```

You can set the “pid” to 0 at the beginning to start from the root node, and after that change it to the item you want to select. Once the “pid” of a media file is entered, the metadata will contain a “VFS_PATH” item, which value is the URL to this media file.

By supplying this URL to SamplePlayer, the playback can be initiated.

```
SamplePlayer <VFS_PATH>
```

Remarks

The arguments of the URLs used in the examples map to the UPnP types as described in the following list.

- **pid**

Describes the ID of the parent node. Maps to ObjectID argument for browse and ContainerID for search request.

- **browse**

Maps to BrowseFlag on browse requests and is ignored on search.

- **search**

Defines the search query, maps to SearchCriteria argument.

- **sort**

Defines the sorting criteria, maps SortCriteria argument.

- **index**

The start index - first item returned. Maps to StartingIndex.

- **count**

Number of result returned. Maps to RequestedCount argument.

Further details on the functions and arguments can be found on:

<http://upnp.org/specs/av/UPnP-av-ContentDirectory-v1-Service.pdf>

14.5.5 Indexing Volumes from a Media Server

Cinemo Media Management allows to index volumes from other Cinemo MM server instances by means of remote indexing. The following prerequisites need to be met for remotely indexing MM servers:

- **Enable journaling**

Indexing of remote servers requires journaling to be enabled on the remote server by means of the option `CINEMO_OPTION_MM_ENABLE_JOURNALING`.

- **Assign server uuids**

Each MM server needs to be assigned a unique and persistent uuid by means of the option *CINEMO_OPTION_MM_SERVER_UUID*.

In order to index a remote volume with a given *volume_id* and a given *server_uuid* the following url shall be used:

```
SampleMM "upnp://192.168.0.1:40000/0/MediaServer/DeviceDesc.xml?cinemo_udn=uuid:<server_uuid>&  
pid=<volume_id>&browse=BrowseDirectChildren"
```

Remarks

The server url including the *cinemo_udn* parameter can be retrieved on the remote server side by calling the [ICinemoMM::GetServerURL\(\)](#) method. The *volume_id* is typically retrieved by registering to volume events (see [Journaling API](#)).

The SampleMMRemotelIndexing sample code provides an example on how to index an entire remote MM server.

Appendix A. Supported Formats

The following formats have been activated for your project. If you are interested in activating additional formats, please contact Cinemo sales at sales@cinemo.com.

Symbols

- ✓ Feature activated in Cinemo SDK release v1.164.0.101454 for the Z101 project.
- (✓) Upcoming feature for the Z101 project, is planned to be activated with a later release.

A.1 File Navigators and Containers

ACTIVE	FILE NAVIGATORS AND CONTAINERS
	AIFF, AIF, AIFC
✓	ASF, WMA, WMV
✓	AVI
✓	AVI DivX Compatibility (up to v6, excluding support for DivX proprietary metadata, DRM and DMF)
	DV, DVSD
✓	Matroska, MKV, MKA, MKS, WEBM/VP8
✓	MPEG, MPEG-1 PS (MPG, DAT, MPE), MPEG-2 PS (MPG, EVO, VOB, VDR, MOD), MPEG-2 TS (TS, TRP, MTS, M2T, M2TS, TOD)
✓	MP4, MP4, M4V, M4A, M4B, 3GP, 3G2, Flash F4V (F4P, F4V, F4A, F4B), Nero Digital, QuickTime (MOV, QT)
	FLV
✓	OGG, OGM, OGV, SPEEX(SPX, SPEEX)
✓	WAV, WAVE
	RA, RV, RM, RMVB

A.2 Playlist Formats

ACTIVE	PLAYLIST FORMATS
✓	M3U
✓	M3U8
✓	PLS
✓	iTunes Playlist (XML format only)
✓	ASX / WAX / WVX / WMX

ACTIVE	PLAYLIST FORMATS
✓	SMIL
✓	RAM
✓	XSPF
✓	KAPSULE
✓	MAGMA
✓	RMP
✓	B4S
✓	WPL
✓	ZPL

A.3 Disc Navigators

ACTIVE	DISC NAVIGATORS
	CDDA (Audio CD)
	DTS-CD
	VCD
	SVCD
	DVD-Video
	AVCHD
	Blu-ray

A.4 Audio Decoders

ACTIVE	AUDIO DECODERS
✓	AAC-LC (MPEG-2)
✓	AAC-LC (MPEG-4) (RealAudio 9)
	AAC Main
	AAC-LTP
✓	AAC-HE V1 (aacPlus)
✓	AAC-HE V2 (Enhanced aacPlus) (RealAudio 10)
✓	AAC-BSAC
✓	AAC-ELD Low Delay
✓	ADPCM: Microsoft
✓	ADPCM: IMA (Microsoft)
✓	ADPCM: IMA (Quicktime)
✓	ADPCM: VOX

ACTIVE	AUDIO DECODERS
✓	ADPCM: G.726
	AMR-NB (Narrow Band)
	AMR-WB (Wide Band)
✓	APE (Monkey's Audio)
	Dolby Digital 2 channel (Requires Dolby Implementation available on system)
	Dolby Digital 5.1 channel (Requires Dolby Implementation available on system)
	Dolby Digital Plus 2 channel (Requires Dolby Implementation available on system)
	Dolby Digital Plus 5.1 channel (Requires Dolby Implementation available on system)
	Dolby Digital Plus 7.1 channel (Requires Dolby Implementation available on system)
	DTS surround (5.1 channel)
	DTS M6 (DTS-HD)
	DTS M8 5.1 (DTS-HD Master Audio)
	DTS M8 (DTS-HD Master Audio 7.1)
✓	FLAC
✓	G.711/PCM A-Law
✓	G.711/PCM μ-Law
✓	G.711/PCM LPCM
	GSM
✓	MPEG-1 Layer 1
✓	MPEG-1 Layer 2
✓	MPEG-1 Layer 3
✓	MPEG-2 Layer 3
	MP3 Pro
	MPEG Multichannel (ISO/IEC 13818-3 MPEG-2 Layer II Multichannel Extensions. Providing support of up to 5.1 audio channels for the MPEG-2 Layer II)
✓	SBC
✓	Speex
✓	Vorbis
✓	WMA 1
✓	WMA 2
✓	WMA 7
✓	WMA 8
✓	WMA 9
✓	WMA 9 Lossless
✓	WMA 9 Voice

ACTIVE	AUDIO DECODERS
✓	WMA 9.1
✓	WMA 10 Pro
✓	WMA 10 Pro LBR
	RealAudio G2/Cook (RealAudio 6)
	RealAudio Lossless Format (RealAudio 10)
	RealAudio Sipro (RealAudio 4/5)
	ATRAC3 (RealAudio 8)
✓	Apple Lossless
	Opus

HUMAX CONFIDENTIAL

A.5 Audio Encoders

ACTIVE	AUDIO ENCODERS
	CDDA Rip&Play
	AAC
	FLAC
	Opus
	ITU G.722 Annex C
	Speex

A.6 Video Decoders

ACTIVE	VIDEO DECODERS
	DV
✓	MJPEG
✓	MPEG-1
✓	MPEG-2 Simple Profile
✓	MPEG-2 Main Profile
✓	MPEG-2 High Profile
✓	MPEG-4 Part 2 Simple Profile
✓	MPEG-4 Part 2 Advanced Simple Profile
✓	MPEG-4 Part 2 DivX Compatibility (up to Version 6, excluding support for DivX proprietary metadata, DRM and DMF)
✓	MPEG-4 Part 2 Xvid
✓	MPEG-4 Part 2 3ivx
✓	MPEG-4 Part 2 Nero Digital
	MS-CRAM (Microsoft Video 1)
	Microsoft MPEG-4 Version
	Microsoft MPEG-4 Version 2
	Microsoft MPEG-4 Version 3
	H.263 / Sorenson Spark (Flash)
✓	H.264 Baseline Profile
✓	H.264 Main Profile
✓	H.264 Extended Profile
✓	H.264 High Profile
✓	H.264 High 4:0:0 Profile
✓	H.264 DivX Compatibility (up to Version 6, excluding support for DivX proprietary metadata, DRM and DMF)

ACTIVE	VIDEO DECODERS
✓	H.264 Nero Digital AVC
	H.265
	H.265 DivX Compatibility
	RAW
	Theora
✓	VP6
✓	VP8 / WebM
✓	VP9
✓	WMV-7
✓	WMV-8
✓	WMV-9
✓	WMV-9 / VC-1 Simple Profile
✓	WMV-9 / VC-1 Main Profile
✓	VC-1 Advanced Profile
	RealVideo 8,9,10

A.7 Image Decoders

ACTIVE	IMAGE DECODERS
✓	BMP
✓	GIF
✓	JPEG
✓	PNG
✓	TIFF
	WBMP
✓	WEBP