Hermes API

Generated by Doxygen 1.8.9.1

Wed May 2 2018 00:53:03

International Distributors

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich, Germany
+49 89 780 2970, info@skyblue.de
www.skyblue.de

In Great Britain:
Zerif Technologies Ltd.
Winnington House, 2 Woodberry Grove
Finchley, London N12 0DR
+44 115 855 7883, info@zerif.co.uk
www.zerif.co.uk

Contents

1 Hermes API **1**

1.1 Introduction . . . 1

1.2 Definitions . . . 1

1.3 Files included in this distribution . . . 2

1.4 Windows Platform SDK & Direct X . . . 3

1.5 Unicode Support . . . 3

1.6 Hermes and multi-threading . . . 3

1.7 Hermes in a Window NT Service . . . 4

1.8 Registering drivers . . . 4

1.9 Hermes API Protection . . . 4

1.10 User Environment Setup Check List . . . 5

1.11 Organization . . . 6

1.12 About time_t . . . 6

2 Module Documentation **7**

2.1 Acquisition . . . 7

2.1.1 Detailed Description . . . 7

2.2 Video . . . 8

2.2.1 Detailed Description . . . 8

2.3 Audio . . . 9

2.3.1 Detailed Description . . . 9

2.4 Data (DAQ) . . . 10

2.4.1 Detailed Description . . . 10

2.5 Image Processing . . . 11

2.5.1 Detailed Description . . . 11

2.6 File Formats . . . 12

2.6.1 Detailed Description . . . 12

2.7 AVI File . . . 13

2.7.1 Detailed Description . . . 13

2.8 MOV File . . . 14

2.9 SEQ File . . . 15

2.9.1 Detailed Description . . . 15
2.10 Image Files . . . 16
2.10.1 Detailed Description . . . 16
2.11 I/O devices . . . 17
2.11.1 Detailed Description . . . 17
2.12 Utilities . . . 18
2.12.1 Detailed Description . . . 19
2.12.2 Function Documentation . . . 19
2.12.2.1 HImageFormatString . . . 19
2.12.2.2 HImageCompressionString . . . 19
2.12.2.3 IsFormatMono . . . 19
2.12.2.4 IsFormatToF . . . 19
2.12.2.5 AbsoluteToRelative . . . 20
2.12.2.6 RelativeToAbsolute . . . 20
2.12.2.7 ComputeTimeDifference . . . 20
2.12.2.8 CompareTimestamps . . . 21
2.12.2.9 AddTimestampOffset . . . 21
2.12.2.10 CreateDirectoryRecursive . . . 21
2.12.2.11 ReadAVIFrameRate . . . 21
2.12.2.12 WriteAVIFrameRate . . . 22
2.12.2.13 IsNvidiaH264Present . . . 22
2.12.2.14 IsNvidiaGPUHEVCPresent . . . 22
2.12.2.15 IsAMDH264Present . . . 22
2.13 Metadata . . . 23
2.13.1 Detailed Description . . . 23
2.14 Generic Metadata . . . 24
2.14.1 Detailed Description . . . 24
2.15 NorPix Metadata . . . 25
2.15.1 Detailed Description . . . 25
2.16 Network . . . 26
2.16.1 Detailed Description . . . 26
2.17 UDP Network . . . 27
2.17.1 Detailed Description . . . 27
2.18 TCP Network . . . 28
2.18.1 Detailed Description . . . 28
2.19 Viewer . . . 29
2.19.1 Detailed Description . . . 29

3 Class Documentation **31**
3.1 CAudioCaptureCallback Class Reference . . . 31

3.1.1 Detailed Description . . . 31
3.1.2 Member Function Documentation . . . 31
3.1.2.1 OnDataReceived . . . 31
3.2 CAudioPlaybackCallback Class Reference . . . 31
3.2.1 Detailed Description . . . 32
3.2.2 Member Function Documentation . . . 32
3.2.2.1 OnReadNewData . . . 32
3.2.2.2 OnAudioSourceEvent . . . 32
3.3 CAudioPlayerCallback Class Reference . . . 32
3.3.1 Detailed Description . . . 32
3.3.2 Member Function Documentation . . . 33
3.3.2.1 OnPlayBuffer . . . 33
3.3.2.2 OnEndPlayback . . . 33
3.4 CAudioRecorderCallback Class Reference . . . 33
3.4.1 Detailed Description . . . 33
3.4.2 Member Function Documentation . . . 33
3.4.2.1 OnBufferReceived . . . 33
3.4.2.2 OnBufferSaved . . . 33
3.5 CCineFormHDParam Struct Reference . . . 34
3.5.1 Detailed Description . . . 34
3.6 CCineFormParam Struct Reference . . . 34
3.6.1 Detailed Description . . . 34
3.7 CCineFormRawParam Struct Reference . . . 35
3.7.1 Detailed Description . . . 35
3.8 CDaqSupportCallback Class Reference . . . 35
3.8.1 Detailed Description . . . 35
3.8.2 Member Function Documentation . . . 35
3.8.2.1 WantContinuousDataRead . . . 35
3.8.2.2 OnDataReceived . . . 36
3.8.2.3 OnDaqDeviceStatusChanged . . . 36
3.8.2.4 WantContinuousDataRead . . . 36
3.8.2.5 OnDataReceived . . . 36
3.8.2.6 OnDaqDeviceStatusChanged . . . 37
3.9 CGPSDataStruct Struct Reference . . . 37
3.9.1 Detailed Description . . . 37
3.10 HAACFile Class Reference . . . 37
3.10.1 Detailed Description . . . 38
3.10.2 Member Function Documentation . . . 38
3.10.2.1 SetErrorOutput . . . 38
3.10.2.2 GetLastError . . . 39

3.10.2.3 Create . . . 39
3.10.2.4 SetAudioParam . . . 39
3.10.2.5 AddData . . . 39
3.10.2.6 EndRecording . . . 40
3.10.2.7 Open . . . 40
3.10.2.8 GetAudioInfo . . . 40
3.10.2.9 GetTotalAudioDatas . . . 40
3.10.2.10 GetTimeStamp . . . 40
3.10.2.11 SetReadingPosition . . . 41
3.10.2.12 GetReadingPosition . . . 41
3.10.2.13 GetPositionByTimeStamp . . . 41
3.10.2.14 ReadAudio . . . 41
3.10.2.15 Close . . . 42
3.11 HAudioData Class Reference . . . 42
3.11.1 Detailed Description . . . 43
3.11.2 Member Function Documentation . . . 43
3.11.2.1 CreateAudioData . . . 43
3.11.2.2 CloneAudioData . . . 43
3.11.2.3 CloneAudioData . . . 43
3.11.2.4 ReallocateAudio . . . 44
3.11.2.5 GetAudioDataBuffer . . . 44
3.11.2.6 GetSampleCount . . . 44
3.11.2.7 GetChannelNum . . . 44
3.11.2.8 GetSamplerate . . . 44
3.11.2.9 GetFormat . . . 45
3.11.2.10 GetAudioBitDepth . . . 45
3.11.2.11 GetAudioBitDepthReal . . . 45
3.11.2.12 GetTimeStamp . . . 45
3.11.2.13 GetTimeStampMS . . . 45
3.11.2.14 GetTimeStampUS . . . 45
3.11.2.15 SetTimeStamp . . . 45
3.12 HAudioDriverASIO Class Reference . . . 46
3.12.1 Detailed Description . . . 47
3.12.2 Constructor & Destructor Documentation . . . 47
3.12.2.1 HAudioDriverASIO . . . 47
3.12.3 Member Function Documentation . . . 48
3.12.3.1 SetErrorOutput . . . 48
3.12.3.2 GetLastError . . . 48
3.12.3.3 GetDriverName . . . 48
3.12.3.4 AudioInputPresent . . . 48

3.12.3.5 GetAudioInputDeviceCount . . . 48
3.12.3.6 GetAudioInputDeviceName . . . 48
3.12.3.7 OpenAudioInputDevice . . . 49
3.12.3.8 CloseAudioInputDevice . . . 49
3.12.3.9 SetAudioInputTimeSourceDLL . . . 49
3.12.3.10 GetAudioInputSampleRateInfoCount . . . 49
3.12.3.11 GetAudioInputSampleRateInfo . . . 50
3.12.3.12 OpenAudioInputHardwareSettingsDlg . . . 50
3.12.3.13 GetAudioInputSampleRate . . . 50
3.12.3.14 GetAudioInputChannelCount . . . 50
3.12.3.15 GetAudioInputChannelInfoByChannelIndex . . . 51
3.12.3.16 StartAudioInputCapture . . . 51
3.12.3.17 StopAudioInputCapture . . . 51
3.12.3.18 IsAudioInputCapturing . . . 52
3.12.3.19 SetCaptureBufferCount . . . 52
3.12.3.20 GetCaptureBufferUsage . . . 52
3.12.3.21 ResetCaptureBufferUsage . . . 52
3.12.3.22 AudioOutputPresent . . . 52
3.12.3.23 GetAudioOutputDeviceCount . . . 52
3.12.3.24 GetAudioOutputDeviceName . . . 53
3.12.3.25 OpenAudioOutputDevice . . . 53
3.12.3.26 CloseAudioOutputDevice . . . 53
3.12.3.27 GetAudioOutputSamplePerBuffer . . . 53
3.12.3.28 GetAudioOutputSampleRateInfoCount . . . 53
3.12.3.29 GetAudioOutputSampleRateInfo . . . 54
3.12.3.30 OpenAudioOutputHardwareSettingsDlg . . . 54
3.12.3.31 GetAudioOutputSampleRate . . . 54
3.12.3.32 GetAudioOutputChannelCount . . . 54
3.12.3.33 GetAudioOutputChannelInfoByChannelIndex . . . 55
3.12.3.34 StartAudioOutputPlayback . . . 55
3.12.3.35 StopAudioOutputPlayback . . . 55
3.12.3.36 IsAudioOutputPlaying . . . 56
3.13 HAudioDriverDirectX Class Reference . . . 56
3.13.1 Detailed Description . . . 58
3.13.2 Constructor & Destructor Documentation . . . 58
3.13.2.1 HAudioDriverDirectX . . . 58
3.13.3 Member Function Documentation . . . 58
3.13.3.1 SetErrorOutput . . . 58
3.13.3.2 GetLastError . . . 58
3.13.3.3 GetDriverName . . . 58

3.13.3.4 AudioInputPresent . . . 58
3.13.3.5 GetAudioInputDeviceCount . . . 59
3.13.3.6 GetAudioInputDeviceName . . . 59
3.13.3.7 OpenAudioInputDevice . . . 59
3.13.3.8 CloseAudioInputDevice . . . 59
3.13.3.9 SetAudioInputTimeSourceDLL . . . 59
3.13.3.10 GetAudioInputFormatCount . . . 60
3.13.3.11 GetAudioInputFormatInfo . . . 60
3.13.3.12 SetAudioInputFormat . . . 60
3.13.3.13 GetAudioInputFormat . . . 61
3.13.3.14 StartAudioInputCapture . . . 61
3.13.3.15 StopAudioInputCapture . . . 61
3.13.3.16 IsAudioInputCapturing . . . 61
3.13.3.17 SetCaptureBufferCount . . . 62
3.13.3.18 GetCaptureBufferUsage . . . 62
3.13.3.19 ResetCaptureBufferUsage . . . 62
3.13.3.20 AudioOutputPresent . . . 62
3.13.3.21 GetAudioOutputDeviceCount . . . 62
3.13.3.22 GetAudioOutputDeviceName . . . 62
3.13.3.23 OpenAudioOutputDevice . . . 63
3.13.3.24 CloseAudioOutputDevice . . . 63
3.13.3.25 GetAudioOutputSampleBufferSize . . . 63
3.13.3.26 GetAudioOutputFormatCount . . . 63
3.13.3.27 GetAudioOutputFormatInfo . . . 64
3.13.3.28 SetAudioOutputFormat . . . 64
3.13.3.29 GetAudioOutputFormat . . . 64
3.13.3.30 StartAudioOutputPlayback . . . 65
3.13.3.31 StopAudioOutputPlayback . . . 65
3.13.3.32 IsAudioOutputPlaying . . . 65
3.13.3.33 SetOutputVolume . . . 65
3.13.3.34 GetOutputVolume . . . 66
3.14 HAudioFile Class Reference . . . 66
3.14.1 Detailed Description . . . 67
3.14.2 Member Function Documentation . . . 68
3.14.2.1 SetErrorOutput . . . 68
3.14.2.2 GetLastError . . . 68
3.14.2.3 GetAudioSamplesPerBuffer . . . 68
3.14.2.4 GetAudioChannelCount . . . 68
3.14.2.5 GetAudioFormat . . . 68
3.14.2.6 GetAudioDepth . . . 68

3.14.2.7 GetAudioDepthReal . . . 68
3.14.2.8 GetSampleRate . . . 69
3.14.2.9 GetAufioFileType . . . 69
3.14.2.10 Create . . . 69
3.14.2.11 Open . . . 69
3.14.2.12 Close . . . 70
3.14.2.13 PreparePlayback . . . 70
3.14.2.14 Write . . . 70
3.14.2.15 Read . . . 70
3.14.2.16 Read . . . 71
3.14.2.17 Read . . . 71
3.14.2.18 SetReadingPosition . . . 71
3.14.2.19 GetAllocatedAudioDatas . . . 71
3.14.2.20 GetRecordingPosition . . . 72
3.14.2.21 GetReadingPosition . . . 72
3.14.2.22 GetPositionByTimeStamp . . . 72
3.14.2.23 GetTimeStamp . . . 72
3.14.2.24 SetStartTimeStamp . . . 73
3.14.2.25 SetPositionByTimeStamp . . . 73
3.15 HAudioPlayer Class Reference . . . 73
3.15.1 Detailed Description . . . 74
3.15.2 Member Function Documentation . . . 74
3.15.2.1 SetErrorOutput . . . 74
3.15.2.2 GetLastError . . . 74
3.15.2.3 SetFileSource . . . 75
3.15.2.4 SetFileSource . . . 76
3.15.2.5 SetFileSource . . . 76
3.15.2.6 SetFileSource . . . 76
3.15.2.7 SetLiveSource . . . 76
3.15.2.8 OpenPlayer . . . 77
3.15.2.9 OpenPlayer . . . 77
3.15.2.10 OpenPlayer . . . 78
3.15.2.11 Play . . . 78
3.15.2.12 Stop . . . 78
3.15.2.13 IsPlaying . . . 78
3.15.2.14 PushLiveAudioData . . . 78
3.15.2.15 MoveTo . . . 79
3.16 HAudioRecorder Class Reference . . . 79
3.16.1 Detailed Description . . . 80
3.16.2 Member Function Documentation . . . 80

3.16.2.1 SetErrorOutput . . . 80
3.16.2.2 GetLastError . . . 81
3.16.2.3 SetCallback . . . 81
3.16.2.4 SetAudioDriver . . . 81
3.16.2.5 SetAudioDriver . . . 81
3.16.2.6 SetAudioDriver . . . 81
3.16.2.7 SetAudioFile . . . 82
3.16.2.8 SetAudioFile . . . 82
3.16.2.9 SetAudioFile . . . 82
3.16.2.10 SetAudioFile . . . 83
3.16.2.11 SetAudioFile . . . 83
3.16.2.12 OpenChannels . . . 83
3.16.2.13 CloseChannels . . . 84
3.16.2.14 AreChannelsOpen . . . 84
3.16.2.15 StartRecording . . . 84
3.16.2.16 StopRecording . . . 84
3.16.2.17 IsRecording . . . 84
3.16.2.18 GetAudioInputLevel . . . 84
3.16.2.19 GetAudioInputLevel . . . 85
3.16.2.20 EnablePassthrough . . . 85
3.16.2.21 SetBufferCount . . . 85
3.16.2.22 GetBufferCount . . . 85
3.16.2.23 GetBufferUsage . . . 85
3.16.2.24 GetMaxBufferUsage . . . 86
3.17 HAutoLocker Class Reference . . . 86
3.17.1 Detailed Description . . . 86
3.18 HAviExporter Class Reference . . . 86
3.18.1 Detailed Description . . . 87
3.18.2 Member Function Documentation . . . 87
3.18.2.1 SetErrorOutput . . . 87
3.18.2.2 GetLastError . . . 87
3.18.2.3 GetVideoCodecList . . . 87
3.18.2.4 SetVideoCodec . . . 88
3.18.2.5 ShowVideoCodecSettings . . . 88
3.18.2.6 GetAudioSourceList . . . 88
3.18.2.7 SetAudioSource . . . 88
3.18.2.8 GetAudioCodecList . . . 89
3.18.2.9 SetAudioCodec . . . 89
3.18.2.10 ShowAudioCodecSettings . . . 89
3.18.2.11 UseSpecialCodec . . . 89

3.18.2.12 ForceFrameRate . . . 90
3.18.2.13 BeginExport . . . 90
3.18.2.14 ExportImage . . . 90
3.18.2.15 EndExport . . . 90
3.18.2.16 IsExporting . . . 91
3.19 HAviFile Class Reference . . . 91
3.19.1 Detailed Description . . . 94
3.19.2 Member Function Documentation . . . 94
3.19.2.1 SetErrorOutput . . . 94
3.19.2.2 GetLastError . . . 94
3.19.2.3 SetCallbacks . . . 94
3.19.2.4 CreateAviFile . . . 95
3.19.2.5 CreateAviFile . . . 95
3.19.2.6 CreateAviFile . . . 95
3.19.2.7 CreateAviFile . . . 96
3.19.2.8 OpenAviFile . . . 97
3.19.2.9 AddFrame . . . 97
3.19.2.10 StopRecording . . . 97
3.19.2.11 CloseAviFile . . . 97
3.19.2.12 GetAviMode . . . 98
3.19.2.13 GetVideoCodecCount . . . 98
3.19.2.14 GetVideoCodecName . . . 98
3.19.2.15 SetVideoCodec . . . 98
3.19.2.16 ShowVideoCodecSettings . . . 98
3.19.2.17 GetAudioSourceCount . . . 99
3.19.2.18 GetAudioSourceName . . . 99
3.19.2.19 SetAudioSource . . . 99
3.19.2.20 GetAudioCodecCount . . . 99
3.19.2.21 GetAudioCodecName . . . 99
3.19.2.22 SetAudioCodec . . . 100
3.19.2.23 ShowAudioCodecSettings . . . 100
3.19.2.24 DetectVideoCodecType . . . 100
3.19.2.25 SetCineFormCodecRawParams . . . 100
3.19.2.26 SetCineFormCodecHDParams . . . 101
3.19.2.27 ForceFrameRate . . . 101
3.19.2.28 IsLiveRecording . . . 101
3.19.2.29 SetLiveRecording . . . 101
3.19.2.30 Play . . . 102
3.19.2.31 Stop . . . 102
3.19.2.32 Pause . . . 102

3.19.2.33 IsPlaying . . . 102
3.19.2.34 IsPaused . . . 102
3.19.2.35 IsStopped . . . 102
3.19.2.36 ReadImage . . . 103
3.19.2.37 GetIndexAt . . . 103
3.19.2.38 SetLooping . . . 103
3.19.2.39 IsLooping . . . 103
3.19.2.40 GetFrameCount . . . 103
3.19.2.41 GetCurrentPosition . . . 104
3.19.2.42 GetPlaybackFrameRate . . . 104
3.19.2.43 GetPlaybackFormat . . . 104
3.19.2.44 SetPlaybackFormat . . . 104
3.19.2.45 GetPlaybackRate . . . 104
3.19.2.46 SetPlaybackRate . . . 104
3.19.2.47 SetTitle . . . 105
3.19.2.48 GetTitle . . . 105
3.19.2.49 SetCopyright . . . 105
3.19.2.50 GetCopyright . . . 105
3.19.2.51 SetCreationDateTime . . . 105
3.19.2.52 GetCreationDateTime . . . 106
3.19.2.53 SetDateTimeOriginal . . . 106
3.19.2.54 GetDateTimeOriginal . . . 106
3.19.2.55 SetAuthor . . . 106
3.19.2.56 GetAuthor . . . 106
3.19.2.57 SetSoftwareName . . . 106
3.19.2.58 GetSoftwareName . . . 107
3.20 HAviFileCallbacks Class Reference . . . 107
3.20.1 Detailed Description . . . 107
3.20.2 Member Function Documentation . . . 107
3.20.2.1 OnAviImageRead . . . 107
3.20.2.2 OnAviEndPlayback . . . 107
3.21 HBitmapOverlay Class Reference . . . 108
3.21.1 Detailed Description . . . 108
3.21.2 Member Function Documentation . . . 109
3.21.2.1 SetErrorOutput . . . 109
3.21.2.2 GetLastError . . . 109
3.21.2.3 SetBitmapFile . . . 109
3.21.2.4 SetBitmapFile . . . 109
3.21.2.5 GetBitmapFile . . . 109
3.21.2.6 SetTransparentcyColor . . . 109

3.21.2.7 SetDisplayColorForBinaryBitmap . . . 110
3.21.2.8 IsBinaryBitmap . . . 110
3.21.2.9 SetOverlayPosition . . . 110
3.21.2.10 SuggestCenterPosition . . . 110
3.21.2.11 SuggestBottomRightPosition . . . 112
3.21.2.12 WriteBitmapOverlay . . . 112
3.22 HDaqDataSet Class Reference . . . 112
3.22.1 Detailed Description . . . 114
3.22.2 Member Function Documentation . . . 114
3.22.2.1 CreateDaqDataSet . . . 114
3.22.2.2 CloneDaqDataSet . . . 114
3.22.2.3 CloneDaqDataSet . . . 114
3.22.2.4 DeleteDaqDataSet . . . 115
3.22.2.5 SetReferenceFrameIndex . . . 115
3.22.2.6 GetSampleCount . . . 115
3.22.2.7 GetChannelNum . . . 115
3.22.2.8 GetSample . . . 115
3.22.2.9 GetTimeStamp . . . 115
3.22.2.10 GetTimeStampMS . . . 116
3.22.2.11 GetTimeStampUS . . . 116
3.22.2.12 SetTimeStamp . . . 116
3.22.2.13 SetDaqDataSet . . . 116
3.22.2.14 SetDummyDataSet . . . 116
3.22.2.15 IsDummyDataSet . . . 117
3.22.2.16 CreateDaqDataSet . . . 117
3.22.2.17 CloneDaqDataSet . . . 117
3.22.2.18 CloneDaqDataSet . . . 117
3.22.2.19 DeleteDaqDataSet . . . 117
3.22.2.20 SetReferenceFrameIndex . . . 118
3.22.2.21 GetSampleCount . . . 118
3.22.2.22 GetChannelNum . . . 118
3.22.2.23 GetSample . . . 118
3.22.2.24 GetTimeStamp . . . 118
3.22.2.25 GetTimeStampMS . . . 119
3.22.2.26 GetTimeStampUS . . . 119
3.22.2.27 SetTimeStamp . . . 119
3.22.2.28 SetDaqDataSet . . . 119
3.22.2.29 SetDummyDataSet . . . 119
3.22.2.30 IsDummyDataSet . . . 120
3.23 HDaqFile Class Reference . . . 120

3.23.1 Detailed Description . . . 123
3.23.2 Member Function Documentation . . . 123
3.23.2.1 SetErrorOutput . . . 123
3.23.2.2 GetLastError . . . 123
3.23.2.3 Create . . . 123
3.23.2.4 Open . . . 123
3.23.2.5 Close . . . 123
3.23.2.6 Write . . . 124
3.23.2.7 Read . . . 124
3.23.2.8 Read . . . 124
3.23.2.9 Read . . . 125
3.23.2.10 GetAllocatedDataSets . . . 125
3.23.2.11 GetRecordingPosition . . . 125
3.23.2.12 GetReadingPosition . . . 125
3.23.2.13 GetSampleRate . . . 125
3.23.2.14 GetRecordingPositionByRefFrame . . . 126
3.23.2.15 GetRecordingPositionByRefTimeStamp . . . 126
3.23.2.16 GetReferenceInfo . . . 126
3.23.2.17 GetPositionAtTime . . . 126
3.23.2.18 GetTimeAtPosition . . . 127
3.23.2.19 SetPlaybackPosition . . . 127
3.23.2.20 GetPlaybackPosition . . . 127
3.23.2.21 StartPlayback . . . 127
3.23.2.22 StopPlayback . . . 128
3.23.2.23 IsPlaying . . . 128
3.23.2.24 ResynchronizePlayback . . . 128
3.23.2.25 SetMetadataSize . . . 128
3.23.2.26 RemoveMetadata . . . 128
3.23.2.27 SetMetadata . . . 129
3.23.2.28 GetMetadataLength . . . 129
3.23.2.29 GetMetadata . . . 129
3.23.2.30 SetErrorOutput . . . 129
3.23.2.31 GetLastError . . . 129
3.23.2.32 Create . . . 129
3.23.2.33 Open . . . 130
3.23.2.34 Close . . . 130
3.23.2.35 Write . . . 130
3.23.2.36 Read . . . 130
3.23.2.37 Read . . . 131
3.23.2.38 Read . . . 131

3.23.2.39 GetAllocatedDataSets . . . 131
3.23.2.40 GetRecordingPosition . . . 132
3.23.2.41 GetReadingPosition . . . 132
3.23.2.42 GetSampleRate . . . 132
3.23.2.43 GetRecordingPositionByRefFrame . . . 132
3.23.2.44 GetRecordingPositionByRefTimeStamp . . . 132
3.23.2.45 GetReferenceInfo . . . 133
3.23.2.46 GetPositionAtTime . . . 133
3.23.2.47 GetTimeAtPosition . . . 133
3.23.2.48 SetPlaybackPosition . . . 134
3.23.2.49 GetPlaybackPosition . . . 134
3.23.2.50 StartPlayback . . . 134
3.23.2.51 StopPlayback . . . 134
3.23.2.52 IsPlaying . . . 134
3.23.2.53 ResynchronizePlayback . . . 135
3.23.2.54 SetMetadataSize . . . 135
3.23.2.55 RemoveMetadata . . . 135
3.23.2.56 SetMetadata . . . 135
3.23.2.57 GetMetadataLength . . . 135
3.23.2.58 GetMetadata . . . 136
3.24 HDaqFileCallbacks Class Reference . . . 136
3.24.1 Detailed Description . . . 136
3.24.2 Member Function Documentation . . . 136
3.24.2.1 OnDaqPlaybackUpdate . . . 136
3.24.2.2 OnDaqPlaybackUpdate . . . 137
3.25 HDeliverCallback Class Reference . . . 137
3.25.1 Detailed Description . . . 137
3.25.2 Member Function Documentation . . . 137
3.25.2.1 OnPrepareCompress . . . 137
3.26 HErrorManager Class Reference . . . 137
3.26.1 Detailed Description . . . 138
3.26.2 Member Function Documentation . . . 138
3.26.2.1 GetErrorString . . . 138
3.27 HFPSMonitor Class Reference . . . 138
3.27.1 Detailed Description . . . 139
3.27.2 Member Function Documentation . . . 139
3.27.2.1 SetErrorOutput . . . 139
3.27.2.2 GetLastError . . . 139
3.27.2.3 StartLiveCheck . . . 139
3.27.2.4 NewLiveFrame . . . 139

3.27.2.5 StartBench . . . 140
3.27.2.6 EndBench . . . 140
3.27.2.7 GetBenchTime . . . 140
3.27.2.8 SetCallbacks . . . 141
3.28 HFPSMonitorCallbacks Class Reference . . . 141
3.28.1 Detailed Description . . . 141
3.28.2 Member Function Documentation . . . 141
3.28.2.1 OnNewFrameRate . . . 141
3.29 HFrameAverage Class Reference . . . 142
3.29.1 Detailed Description . . . 142
3.29.2 Member Function Documentation . . . 143
3.29.2.1 SetErrorOutput . . . 143
3.29.2.2 GetLastError . . . 143
3.29.2.3 AllocateAverageBuffer . . . 143
3.29.2.4 FreeAverageBuffer . . . 143
3.29.2.5 StandardFrameAverage . . . 143
3.29.2.6 RollingFrameAverage . . . 144
3.29.2.7 SetErrorOutput . . . 144
3.29.2.8 GetLastError . . . 144
3.29.2.9 AllocateAverageBuffer . . . 144
3.29.2.10 FreeAverageBuffer . . . 144
3.29.2.11 StandardFrameAverage . . . 145
3.29.2.12 RollingFrameAverage . . . 145
3.30 HGPUDeviceManager Class Reference . . . 145
3.30.1 Member Function Documentation . . . 147
3.30.1.1 SetErrorOutput . . . 147
3.30.1.2 GetLastError . . . 147
3.30.1.3 GetIntelGPUDeviceInfo . . . 147
3.30.1.4 GetNvidiaGPUDeviceInfo . . . 147
3.30.1.5 EnableNvidiaOpenCLEncoding . . . 147
3.30.1.6 IsNvidiaOpenCLEncodingEnabled . . . 147
3.30.1.7 DisableNvidiaOpenCLEncoding . . . 147
3.30.1.8 EnableNvidiaCudaEncoding . . . 148
3.30.1.9 IsNvidiaCudaEncodingEnabled . . . 148
3.30.1.10 DisableNvidiaCudaEncoding . . . 148
3.30.1.11 GetAMDGPUDeviceInfo . . . 148
3.30.1.12 EnableAMDOpenCLEncoding . . . 148
3.30.1.13 IsAMDOpenCLEncodingEnabled . . . 148
3.30.1.14 DisableAMDOpenCLEncoding . . . 149
3.31 HGrabCallbacks Class Reference . . . 149

3.31.1 Detailed Description . . . 149
3.31.2 Member Function Documentation . . . 150
3.31.2.1 OnBeforeImageReceived . . . 150
3.31.2.2 OnImageReceived . . . 150
3.31.2.3 OnImageReleased . . . 150
3.31.2.4 OnBeforeQueuingGrab . . . 151
3.31.2.5 OnAfterQueuingGrab . . . 151
3.31.2.6 OnBeforeStartStreaming . . . 151
3.31.2.7 OnAfterStartStreaming . . . 151
3.31.2.8 OnBeforeStopStreaming . . . 151
3.31.2.9 OnAfterStopStreaming . . . 152
3.31.2.10 OnNotification . . . 152
3.31.2.11 OnGPSSentencesReceived . . . 152
3.32 HGrabModule Class Reference . . . 152
3.32.1 Detailed Description . . . 158
3.32.2 Member Function Documentation . . . 158
3.32.2.1 SetErrorOutput . . . 158
3.32.2.2 GetLastError . . . 158
3.32.2.3 GetDriverCount . . . 158
3.32.2.4 GetDriverInfo . . . 159
3.32.2.5 GetDriverInfo . . . 159
3.32.2.6 Load . . . 159
3.32.2.7 Load . . . 160
3.32.2.8 Unload . . . 160
3.32.2.9 GetCurrentGrabber . . . 160
3.32.2.10 ExportGrabberSettings . . . 160
3.32.2.11 StartStreaming . . . 161
3.32.2.12 StopStreaming . . . 161
3.32.2.13 IsStreaming . . . 161
3.32.2.14 SetCallbacks . . . 161
3.32.2.15 ShowProperties . . . 162
3.32.2.16 GetGrabberInfoCount . . . 162
3.32.2.17 GetGrabberInfoName . . . 162
3.32.2.18 GetGrabberInfoValueType . . . 163
3.32.2.19 GetGrabberInfoValueLong . . . 163
3.32.2.20 GetGrabberInfoValueDouble . . . 163
3.32.2.21 GetGrabberInfoValueString . . . 163
3.32.2.22 SetROI . . . 164
3.32.2.23 GetROI . . . 164
3.32.2.24 SetBinning . . . 164

3.32.2.25 GetBinning . . . 165
3.32.2.26 GetMaxResolution . . . 165
3.32.2.27 GetSettingsCount . . . 165
3.32.2.28 GetSettingsCaps . . . 165
3.32.2.29 GetValuesCount . . . 166
3.32.2.30 GetValuesCaps . . . 166
3.32.2.31 GetCurrentValues . . . 166
3.32.2.32 SetCurrentValues . . . 166
3.32.2.33 GetAdjustmentCount . . . 167
3.32.2.34 GetAdjustmentCaps . . . 167
3.32.2.35 GetIntegerAdjustment . . . 168
3.32.2.36 SetIntegerAdjustment . . . 168
3.32.2.37 GetDoubleAdjustment . . . 168
3.32.2.38 SetDoubleAdjustment . . . 169
3.32.2.39 GetAutomaticAdjustment . . . 169
3.32.2.40 SetAutomaticAdjustment . . . 169
3.32.2.41 OneShotAdjustment . . . 169
3.32.2.42 GetImageWidth . . . 170
3.32.2.43 GetImageHeight . . . 170
3.32.2.44 GetImageBitDepth . . . 170
3.32.2.45 GetImageBitDepthReal . . . 170
3.32.2.46 GetImageSizeBytes . . . 170
3.32.2.47 GetImageFormat . . . 171
3.32.2.48 IsGPSSupported . . . 171
3.32.2.49 GetGPSData . . . 171
3.32.2.50 StopReadingGPSData . . . 171
3.32.2.51 IsCOMPortAvailable . . . 171
3.32.2.52 Transmit . . . 172
3.32.2.53 Receive . . . 172
3.32.2.54 GetEndChar . . . 172
3.32.2.55 GetTextMode . . . 173
3.32.2.56 GetProgFolder . . . 173
3.32.2.57 GetSavedProgFolder . . . 173
3.32.2.58 GetCommunicationType . . . 173
3.32.2.59 IsSupportAdvanceSettings . . . 174
3.32.2.60 DoModalAdvanceSettingsWindow . . . 174
3.32.2.61 CreateAdvanceSettingsWindow . . . 174
3.32.2.62 ReSizeAdvanceSettingsWindow . . . 174
3.32.2.63 ShowAdvanceSettingsWindow . . . 175
3.32.2.64 CloseAdvanceSettingsWindow . . . 175

3.32.2.65 GetCategoriesCount . . . 175
3.32.2.66 GetCategoriesCap . . . 176
3.32.2.67 GetFeatureCount . . . 176
3.32.2.68 GetFeatureCap . . . 176
3.32.2.69 Accessible . . . 176
3.32.2.70 GetEnumFeatureValueCount . . . 177
3.32.2.71 GetEnumFeatureValueCap . . . 177
3.32.2.72 GetEnumFeatureValue . . . 177
3.32.2.73 SetEnumFeatureValue . . . 178
3.32.2.74 GetIntegerFeatureCap . . . 178
3.32.2.75 GetIntegerFeatureValue . . . 178
3.32.2.76 SetIntegerFeatureValue . . . 178
3.32.2.77 GetDoubleFeatureCap . . . 179
3.32.2.78 GetDoubleFeatureValue . . . 179
3.32.2.79 SetDoubleFeatureValue . . . 179
3.32.2.80 GetBoolFeatureValue . . . 179
3.32.2.81 SetBoolFeatureValue . . . 180
3.32.2.82 ExecuteCommandFeature . . . 180
3.32.2.83 GetIODriver . . . 180
3.32.2.84 StartQueryMode . . . 180
3.32.2.85 GetGrabberCount . . . 181
3.32.2.86 GetGrabberName . . . 181
3.32.2.87 SetDefaultGrabber . . . 181
3.32.2.88 EndQueryMode . . . 181
3.32.2.89 SetBufferCount . . . 182
3.32.2.90 GetBufferCount . . . 182
3.32.2.91 GetBufferUsage . . . 182
3.32.2.92 GetMaxBufferUsage . . . 182
3.32.2.93 GetTotalBufferCount . . . 182
3.32.2.94 IsSupportPanTilt . . . 183
3.32.2.95 IsSupportZoom . . . 183
3.32.2.96 GetPanTiltXRange . . . 183
3.32.2.97 GetZoomRange . . . 183
3.32.2.98 SetPanTilt . . . 183
3.32.2.99 SetZoom . . . 184
3.32.2.100 GetPanTilt . . . 184
3.32.2.101 GetZoom . . . 184
3.33 HGrayConverter Class Reference . . . 184
3.33.1 Detailed Description . . . 185
3.33.2 Member Function Documentation . . . 185

3.33.2.1 SetErrorOutput . . . 185
3.33.2.2 GetLastError . . . 185
3.33.2.3 GrayToMono8 . . . 185
3.33.2.4 CheckGrayConvertionImageFormat . . . 185
3.33.2.5 SetErrorOutput . . . 186
3.33.2.6 GetLastError . . . 186
3.33.2.7 GrayToMono8 . . . 186
3.33.2.8 CheckGrayConvertionImageFormat . . . 186
3.34 HImage Class Reference . . . 186
3.34.1 Detailed Description . . . 188
3.34.2 Member Function Documentation . . . 188
3.34.2.1 CreateHImage . . . 188
3.34.2.2 CloneHImage . . . 189
3.34.2.3 CloneHImage . . . 189
3.34.2.4 DeleteHImage . . . 189
3.34.2.5 GetImageWidth . . . 189
3.34.2.6 GetImageHeight . . . 189
3.34.2.7 GetImageBitDepth . . . 190
3.34.2.8 GetImageBitDepthReal . . . 190
3.34.2.9 GetImageSizeBytes . . . 190
3.34.2.10 GetImageFormat . . . 190
3.34.2.11 GetRawImageData . . . 190
3.34.2.12 GetRedChannel . . . 190
3.34.2.13 GetGreenChannel . . . 191
3.34.2.14 GetBlueChannel . . . 191
3.34.2.15 ReallocateImage . . . 191
3.34.2.16 ReallocateCompressedImage . . . 191
3.34.2.17 HasImageDecoded . . . 191
3.34.2.18 GetTimestamp . . . 192
3.34.2.19 GetTimestampMS . . . 192
3.34.2.20 GetTimestampUS . . . 192
3.34.2.21 SetTimestamp . . . 192
3.34.2.22 GetMetadataCount . . . 192
3.34.2.23 GetMetadata . . . 193
3.34.2.24 GetSpecificMetadataCount . . . 193
3.34.2.25 GetSpecificMetadata . . . 193
3.34.2.26 AddMetadata . . . 193
3.35 HImageColorProcessing Class Reference . . . 193
3.35.1 Detailed Description . . . 196
3.35.2 Member Function Documentation . . . 196

3.35.2.1 SetErrorOutput . . . 196
3.35.2.2 GetLastError . . . 197
3.35.2.3 ApplyColorProcessing . . . 197
3.35.2.4 ApplyColorProcessing . . . 197
3.35.2.5 EnableBayerConversion . . . 197
3.35.2.6 IsBayerConversionEnabled . . . 197
3.35.2.7 SetPatternOrigin . . . 198
3.35.2.8 GetPatternOrigin . . . 199
3.35.2.9 SetCalcMode . . . 199
3.35.2.10 GetCalcMode . . . 199
3.35.2.11 SetSampleFactor . . . 199
3.35.2.12 GetSampleFactor . . . 199
3.35.2.13 EnableBGRxConversion . . . 199
3.35.2.14 IsBGRxConversionEnabled . . . 200
3.35.2.15 EnableColorProcessingHQConversion . . . 200
3.35.2.16 IsColorProcessingHQConversionEnabled . . . 200
3.35.2.17 IsMatrixColorCorrectionEnabled . . . 200
3.35.2.18 EnableMatrixColorCorrection . . . 200
3.35.2.19 GetCorrectionMatrix . . . 201
3.35.2.20 SetCorrectionMatrix . . . 201
3.35.2.21 IsAutoColorBalanceEnabled . . . 202
3.35.2.22 EnableAutoColorBalance . . . 202
3.35.2.23 GetManualColorBalance . . . 202
3.35.2.24 SetManualColorBalance . . . 202
3.35.2.25 GetAutoColorBalanceAlgorithm . . . 202
3.35.2.26 SetAutoColorBalanceAlgorithm . . . 203
3.35.2.27 SetAutoWhiteBalanceROI . . . 203
3.35.2.28 GetAutoWhiteBalanceROI . . . 203
3.35.2.29 GetASCSettingsParam . . . 203
3.35.2.30 SetASCSettingsParam . . . 204
3.35.2.31 IsLUTProcessEnabled . . . 204
3.35.2.32 EnableLUTProcess . . . 204
3.35.2.33 Is3DLookupTable . . . 204
3.35.2.34 Enable3DLookupTable . . . 204
3.35.2.35 SetCacheLUTFileName . . . 205
3.35.2.36 Get1DSupportThirdPartLutFileCount . . . 205
3.35.2.37 Get1DSupportThirdPartLutFileFormatInfo . . . 205
3.35.2.38 Get3DSupportThirdPartLutFileCount . . . 205
3.35.2.39 Get3DSupportThirdPartLutFileFormatInfo . . . 206
3.35.2.40 AddThirdPartLutFile . . . 206

3.35.2.41 RemoveThirdPartLutFile . . . 206
3.35.2.42 SetColorClipLevel . . . 206
3.35.2.43 GetColorClipLevel . . . 207
3.35.2.44 SetMonoPseudoColorMode . . . 207
3.35.2.45 GetMonoPseudoColorMode . . . 207
3.35.2.46 Get1DPreDefinedLUTCount . . . 207
3.35.2.47 Get1DPreDefinedLUT . . . 207
3.35.2.48 Load1DPreDefinedLUT . . . 208
3.35.2.49 LoadNpxTextLutFile . . . 208
3.35.2.50 ExportLUTToTextFile . . . 208
3.35.2.51 EnableMonochromeConversion . . . 209
3.35.2.52 IsMonochromeConversionEnabled . . . 209
3.35.2.53 Get1DLut . . . 209
3.36 HImageConverter Class Reference . . . 209
3.36.1 Detailed Description . . . 210
3.36.2 Member Function Documentation . . . 210
3.36.2.1 SetErrorOutput . . . 210
3.36.2.2 GetLastError . . . 210
3.36.2.3 ConvertToBGR . . . 210
3.36.2.4 ConvertToBGRx . . . 211
3.36.2.5 ConvertToMono8 . . . 211
3.37 HImageExporter Class Reference . . . 211
3.37.1 Detailed Description . . . 213
3.37.2 Member Function Documentation . . . 213
3.37.2.1 SetErrorOutput . . . 213
3.37.2.2 GetLastError . . . 213
3.37.2.3 BeginExportBmp . . . 214
3.37.2.4 BeginExportJpeg . . . 214
3.37.2.5 BeginExportTiff . . . 214
3.37.2.6 BeginExportPng . . . 215
3.37.2.7 BeginExportJpeg2k . . . 215
3.37.2.8 BeginExportFit . . . 215
3.37.2.9 BeginExportDpx . . . 216
3.37.2.10 BeginExportDng . . . 216
3.37.2.11 BeginExportJpegXR . . . 217
3.37.2.12 ExportImage . . . 217
3.37.2.13 EndExport . . . 217
3.37.2.14 BeginExportBmpTo . . . 218
3.37.2.15 BeginExportJpegTo . . . 218
3.37.2.16 BeginExportTiffTo . . . 218

3.37.2.17 BeginExportPngTo . . . 218
3.37.2.18 BeginExportJpeg2kTo . . . 218
3.37.2.19 BeginExportFitTo . . . 219
3.37.2.20 BeginExportDpxTo . . . 219
3.37.2.21 BeginExportDngTo . . . 219
3.37.2.22 BeginExportJpegXRTo . . . 219
3.37.2.23 ExportImageTo . . . 220
3.37.2.24 EndExportTo . . . 220
3.37.2.25 SetJPEGQuality . . . 220
3.37.2.26 GetJPEGQuality . . . 221
3.37.2.27 SetJpegEncoderType . . . 221
3.37.2.28 SetJpegGPUEncoderType . . . 221
3.37.2.29 SetJpegGPUCompressionBayerPatternType . . . 221
3.37.2.30 EnableGPUJpegCompressionBayerInterp . . . 221
3.37.2.31 EnableGPUJpegCompressionColorprcessingLUT . . . 222
3.37.2.32 UpdateGPUJpegCompressionLUT . . . 222
3.37.2.33 AddJPEGMetadata . . . 222
3.37.2.34 EnableTIFFMetadata . . . 222
3.37.2.35 GetDateTimeFormat . . . 223
3.37.2.36 SetDateTimeFormat . . . 223
3.37.2.37 SetSeparator . . . 223
3.38 HImageMerge Class Reference . . . 223
3.38.1 Detailed Description . . . 224
3.38.2 Member Function Documentation . . . 224
3.38.2.1 SetErrorOutput . . . 224
3.38.2.2 GetLastError . . . 224
3.38.2.3 SetMergeMode . . . 224
3.38.2.4 GetMergeMode . . . 224
3.38.2.5 ForceOutputBRG24 . . . 224
3.38.2.6 CheckImageMergeFormat . . . 224
3.38.2.7 Merge . . . 225
3.38.2.8 MultipleMerge . . . 225
3.38.2.9 Stitch . . . 225
3.39 HImageMirror Class Reference . . . 226
3.39.1 Detailed Description . . . 226
3.39.2 Member Function Documentation . . . 226
3.39.2.1 SetErrorOutput . . . 226
3.39.2.2 GetLastError . . . 226
3.39.2.3 SetMirrorMode . . . 226
3.39.2.4 GetMirrorMode . . . 227

3.39.2.5 CheckImageMirrorFormat . . . 227
3.39.2.6 Mirror . . . 227
3.40 HImageOverlay Class Reference . . . 227
3.40.1 Detailed Description . . . 229
3.40.2 Member Function Documentation . . . 229
3.40.2.1 SetErrorOutput . . . 229
3.40.2.2 GetLastError . . . 229
3.40.2.3 SetOverlayColor . . . 229
3.40.2.4 GetOverlayColor . . . 230
3.40.2.5 SetTextBackgroundColor . . . 230
3.40.2.6 GetTextBackgroundColor . . . 230
3.40.2.7 SetAlpha . . . 230
3.40.2.8 GetAlpha . . . 230
3.40.2.9 SetTextOverlayAlign . . . 231
3.40.2.10 SetTextOverlayFont . . . 231
3.40.2.11 SetTextOverlaySize . . . 231
3.40.2.12 SetCrosshairThickness . . . 231
3.40.2.13 GetCrosshairThickness . . . 232
3.40.2.14 SetLineThickness . . . 232
3.40.2.15 GetLineThickness . . . 232
3.40.2.16 SetLineStyle . . . 232
3.40.2.17 SetLineColor . . . 232
3.40.2.18 GetLineColor . . . 233
3.40.2.19 GetPolygonPixelCount . . . 233
3.40.2.20 BeginDrawing . . . 233
3.40.2.21 EndDrawing . . . 233
3.40.2.22 WriteAlignedTextOverlay . . . 233
3.40.2.23 WriteAlignedMultilinesTextOverlay . . . 234
3.40.2.24 WriteTextOverlay . . . 234
3.40.2.25 WriteMultilineTextOverlay . . . 234
3.40.2.26 WriteLineOverlay . . . 235
3.40.2.27 WriteFlatEndsLineOverlay . . . 235
3.40.2.28 WriteRoundEndsLineOverlay . . . 235
3.40.2.29 WriteCrosshair . . . 235
3.40.2.30 WriteArrowOverlay . . . 236
3.40.2.31 WriteRectOverlay . . . 236
3.40.2.32 WriteFilledRectOverlay . . . 236
3.40.2.33 WriteEllipseOverlay . . . 237
3.40.2.34 WriteFilledEllipseOverlay . . . 237
3.40.2.35 WriteCircleOverlay . . . 237

3.40.2.36 WriteAngleOverlay . . . 237
3.40.2.37 WritePolyOverlay . . . 238
3.40.2.38 WriteFilledPolyOverlay . . . 238
3.40.2.39 WriteBezierOverlay . . . 238
3.40.2.40 WriteCustomOverlay . . . 239
3.41 HImageResize Class Reference . . . 239
3.41.1 Detailed Description . . . 240
3.41.2 Member Function Documentation . . . 240
3.41.2.1 SetErrorOutput . . . 240
3.41.2.2 GetLastError . . . 240
3.41.2.3 SetSourceRoi . . . 240
3.41.2.4 GetSourceRoi . . . 240
3.41.2.5 SetDestinationSize . . . 240
3.41.2.6 GetDestinationSize . . . 241
3.41.2.7 SetKeepImageRatio . . . 241
3.41.2.8 IsKeepImageRatio . . . 241
3.41.2.9 SetInterpolationMode . . . 241
3.41.2.10 GetInterpolationMode . . . 241
3.41.2.11 CheckImageResizeFormat . . . 241
3.41.2.12 Resize . . . 242
3.41.2.13 CheckImageReSampleFormat . . . 242
3.41.2.14 ReSample . . . 242
3.42 HIODriver Class Reference . . . 242
3.42.1 Detailed Description . . . 245
3.42.2 Constructor & Destructor Documentation . . . 245
3.42.2.1 HIODriver . . . 245
3.42.3 Member Function Documentation . . . 245
3.42.3.1 GetDriverName . . . 245
3.42.3.2 InputDevicePresent . . . 245
3.42.3.3 GetInputDeviceCount . . . 245
3.42.3.4 GetInputDeviceName . . . 246
3.42.3.5 InitializeDeviceInput . . . 246
3.42.3.6 UnInitializeDeviceInput . . . 246
3.42.3.7 GetDeviceInputLineCount . . . 246
3.42.3.8 GetDeviceInputLineInfo . . . 247
3.42.3.9 GetPollingDelay . . . 247
3.42.3.10 GetCurrentLevel . . . 247
3.42.3.11 Monitor . . . 247
3.42.3.12 OutputDevicePresent . . . 248
3.42.3.13 GetOutputDeviceCount . . . 248

3.42.3.14 GetOutputDeviceName . . . 248
3.42.3.15 InitializeDeviceOutput . . . 248
3.42.3.16 UnInitializeDeviceOutput . . . 249
3.42.3.17 GetDeviceOutputLineCount . . . 249
3.42.3.18 GetDeviceOutputLineInfo . . . 249
3.42.3.19 GetOutputLevel . . . 249
3.42.3.20 SetOutputLevel . . . 250
3.42.3.21 PulseGeneratorPresent . . . 250
3.42.3.22 InitializePulseGenerator . . . 250
3.42.3.23 UnInitializePulseGenerator . . . 250
3.42.3.24 StartPulseGenerator . . . 250
3.42.3.25 StopPulseGenerator . . . 251
3.42.3.26 DaqPresent . . . 251
3.42.3.27 GetDaqDeviceCount . . . 251
3.42.3.28 GetDaqDeviceName . . . 251
3.42.3.29 InitializeDaqDevice . . . 251
3.42.3.30 UnInitializeDaqDevice . . . 252
3.42.3.31 SetTimeSourceDLL . . . 252
3.42.3.32 GetDaqDeviceLineCount . . . 252
3.42.3.33 GetDaqDeviceLineInfo . . . 252
3.42.3.34 GetDaqDeviceLineRangeSettingsCount . . . 252
3.42.3.35 GetDaqDeviceLineRangeSettingsInfo . . . 253
3.42.3.36 GetMaxSampleRate . . . 253
3.42.3.37 StartCapture . . . 253
3.42.3.38 StopCapture . . . 254
3.42.3.39 GetSample . . . 254
3.42.3.40 GetDaqDeviceParametersCount . . . 254
3.42.3.41 GetDaqDeviceParametersInfo . . . 255
3.42.3.42 GetDaqDeviceEnumParameterItemCount . . . 255
3.42.3.43 GetDaqDeviceEnumParameterItemInfo . . . 255
3.42.3.44 GetDaqDeviceFloatParameterItemInfo . . . 255
3.42.3.45 GetDaqDeviceIntParameterItemInfo . . . 256
3.42.3.46 GetDaqDeviceEnumParameterValue . . . 256
3.42.3.47 SetDaqDeviceEnumParameterValue . . . 256
3.42.3.48 GetDaqDeviceFloatParameterValue . . . 257
3.42.3.49 SetDaqDeviceFloatParameterValue . . . 257
3.42.3.50 GetDaqDeviceIntParameterValue . . . 257
3.42.3.51 SetDaqDeviceIntParameterValue . . . 257
3.43 HItem Class Reference . . . 258
3.43.1 Detailed Description . . . 258

3.43.2 Member Function Documentation . . . 258
3.43.2.1 GetID . . . 258
3.43.2.2 GetDescription . . . 258
3.44 HJpegCompression Class Reference . . . 258
3.44.1 Detailed Description . . . 259
3.44.2 Member Function Documentation . . . 259
3.44.2.1 SetErrorOutput . . . 259
3.44.2.2 GetLastError . . . 259
3.44.2.3 IsFormatSupported . . . 259
3.44.2.4 Encode . . . 259
3.44.2.5 Decode . . . 260
3.45 HLineScanManager Class Reference . . . 260
3.45.1 Detailed Description . . . 261
3.45.2 Member Function Documentation . . . 262
3.45.2.1 Configure . . . 262
3.45.2.2 SetLineCount . . . 263
3.45.2.3 SetDirection . . . 263
3.45.2.4 PushImageToQueue . . . 263
3.45.2.5 GetLiveImage . . . 263
3.45.2.6 GetImageFromSeq . . . 263
3.45.2.7 ConfigureImageExport . . . 264
3.45.2.8 ConfigureAVIExport . . . 264
3.45.2.9 ConfigureMOVExport . . . 264
3.45.2.10 Export . . . 265
3.45.2.11 GetCurrentExportedFrame . . . 265
3.45.2.12 GetTimeStampImages . . . 265
3.45.2.13 Configure . . . 265
3.45.2.14 SetLineCount . . . 266
3.45.2.15 SetDirection . . . 266
3.45.2.16 PushImageToQueue . . . 266
3.45.2.17 GetLiveImage . . . 266
3.45.2.18 GetImageFromSeq . . . 266
3.45.2.19 ConfigureImageExport . . . 267
3.45.2.20 ConfigureAVIExport . . . 267
3.45.2.21 ConfigureMOVExport . . . 267
3.45.2.22 Export . . . 268
3.45.2.23 GetCurrentExportedFrame . . . 268
3.45.2.24 GetTimeStampImages . . . 268
3.46 HLocker Class Reference . . . 268
3.46.1 Detailed Description . . . 269

3.46.2 Member Function Documentation . . . 269
3.46.2.1 TryToLock . . . 269
3.47 HMetadata Class Reference . . . 269
3.47.1 Detailed Description . . . 270
3.47.2 Constructor & Destructor Documentation . . . 270
3.47.2.1 HMetadata . . . 270
3.47.2.2 HMetadata . . . 271
3.47.2.3 HMetadata . . . 271
3.47.3 Member Function Documentation . . . 271
3.47.3.1 operator= . . . 271
3.47.3.2 BuildFrom . . . 271
3.47.3.3 GetUID . . . 271
3.47.3.4 GetDataSizeBytes . . . 271
3.47.3.5 GetData . . . 272
3.47.3.6 SetMetadataString . . . 272
3.47.3.7 GetMetadataString . . . 272
3.47.3.8 GetPacketSize . . . 272
3.47.3.9 WritePacketTo . . . 272
3.48 HMetadataGenericBinary Class Reference . . . 273
3.48.1 Detailed Description . . . 273
3.48.2 Constructor & Destructor Documentation . . . 273
3.48.2.1 HMetadataGenericBinary . . . 273
3.48.2.2 HMetadataGenericBinary . . . 273
3.48.2.3 HMetadataGenericBinary . . . 273
3.49 HMetadataGenericBool Class Reference . . . 274
3.49.1 Detailed Description . . . 274
3.49.2 Constructor & Destructor Documentation . . . 274
3.49.2.1 HMetadataGenericBool . . . 274
3.49.2.2 HMetadataGenericBool . . . 274
3.49.2.3 HMetadataGenericBool . . . 275
3.49.3 Member Function Documentation . . . 275
3.49.3.1 BuildMetadataString . . . 275
3.50 HMetadataGenericByte Class Reference . . . 275
3.50.1 Detailed Description . . . 275
3.50.2 Constructor & Destructor Documentation . . . 276
3.50.2.1 HMetadataGenericByte . . . 276
3.50.2.2 HMetadataGenericByte . . . 276
3.50.2.3 HMetadataGenericByte . . . 276
3.50.3 Member Function Documentation . . . 276
3.50.3.1 BuildMetadataString . . . 276

3.51 HMetadataGenericDouble Class Reference . . . 276
3.51.1 Detailed Description . . . 277
3.51.2 Constructor & Destructor Documentation . . . 277
3.51.2.1 HMetadataGenericDouble . . . 277
3.51.2.2 HMetadataGenericDouble . . . 277
3.51.2.3 HMetadataGenericDouble . . . 277
3.51.3 Member Function Documentation . . . 277
3.51.3.1 BuildMetadataString . . . 277
3.52 HMetadataGenericInt Class Reference . . . 278
3.52.1 Detailed Description . . . 278
3.52.2 Constructor & Destructor Documentation . . . 278
3.52.2.1 HMetadataGenericInt . . . 278
3.52.2.2 HMetadataGenericInt . . . 278
3.52.2.3 HMetadataGenericInt . . . 279
3.52.3 Member Function Documentation . . . 279
3.52.3.1 BuildMetadataString . . . 279
3.53 HMetadataGenericInt64 Class Reference . . . 279
3.53.1 Detailed Description . . . 279
3.53.2 Constructor & Destructor Documentation . . . 280
3.53.2.1 HMetadataGenericInt64 . . . 280
3.53.2.2 HMetadataGenericInt64 . . . 280
3.53.2.3 HMetadataGenericInt64 . . . 280
3.53.3 Member Function Documentation . . . 280
3.53.3.1 BuildMetadataString . . . 280
3.54 HMetadataGenericShort Class Reference . . . 280
3.54.1 Detailed Description . . . 281
3.54.2 Constructor & Destructor Documentation . . . 281
3.54.2.1 HMetadataGenericShort . . . 281
3.54.2.2 HMetadataGenericShort . . . 281
3.54.2.3 HMetadataGenericShort . . . 281
3.54.3 Member Function Documentation . . . 281
3.54.3.1 BuildMetadataString . . . 281
3.55 HMetadataGenericString Class Reference . . . 282
3.55.1 Detailed Description . . . 282
3.55.2 Constructor & Destructor Documentation . . . 282
3.55.2.1 HMetadataGenericString . . . 282
3.55.2.2 HMetadataGenericString . . . 282
3.55.2.3 HMetadataGenericString . . . 283
3.55.3 Member Function Documentation . . . 283
3.55.3.1 BuildMetadataString . . . 283

3.56 HMetadataGenericUInt Class Reference . . . 283
3.56.1 Detailed Description . . . 283
3.56.2 Constructor & Destructor Documentation . . . 284
3.56.2.1 HMetadataGenericUInt . . . 284
3.56.2.2 HMetadataGenericUInt . . . 284
3.56.2.3 HMetadataGenericUInt . . . 284
3.56.3 Member Function Documentation . . . 284
3.56.3.1 BuildMetadataString . . . 284
3.57 HMetadataGenericUInt64 Class Reference . . . 284
3.57.1 Detailed Description . . . 285
3.57.2 Constructor & Destructor Documentation . . . 285
3.57.2.1 HMetadataGenericUInt64 . . . 285
3.57.2.2 HMetadataGenericUInt64 . . . 285
3.57.2.3 HMetadataGenericUInt64 . . . 285
3.57.3 Member Function Documentation . . . 285
3.57.3.1 BuildMetadataString . . . 285
3.58 HMetadataGenericUShort Class Reference . . . 286
3.58.1 Detailed Description . . . 286
3.58.2 Constructor & Destructor Documentation . . . 286
3.58.2.1 HMetadataGenericUShort . . . 286
3.58.2.2 HMetadataGenericUShort . . . 286
3.58.2.3 HMetadataGenericUShort . . . 287
3.58.3 Member Function Documentation . . . 287
3.58.3.1 BuildMetadataString . . . 287
3.59 HMetadataInfo Class Reference . . . 287
3.59.1 Detailed Description . . . 288
3.59.2 Constructor & Destructor Documentation . . . 288
3.59.2.1 HMetadataInfo . . . 288
3.59.2.2 HMetadataInfo . . . 288
3.59.3 Member Function Documentation . . . 288
3.59.3.1 GetPacketSize . . . 288
3.59.3.2 WritePacketTo . . . 288
3.60 HMetadataManager Class Reference . . . 289
3.60.1 Detailed Description . . . 289
3.60.2 Member Function Documentation . . . 289
3.60.2.1 RegisterMetadataType . . . 289
3.60.2.2 RegisterGenericTypes . . . 290
3.60.2.3 RegisterNorPixTypes . . . 290
3.60.2.4 GetMetadataTypeCount . . . 290
3.60.2.5 GetMetadataTypeByIndex . . . 290

3.60.2.6 GetMetadataTypeByID . . . 290
3.60.2.7 RebuildMetadataString . . . 290
3.61 HMetadataNorPixCameraInfo Class Reference . . . 291
3.61.1 Detailed Description . . . 291
3.61.2 Constructor & Destructor Documentation . . . 292
3.61.2.1 HMetadataNorPixCameraInfo . . . 292
3.61.2.2 HMetadataNorPixCameraInfo . . . 292
3.61.3 Member Function Documentation . . . 292
3.61.3.1 BuildMetadataString . . . 292
3.61.3.2 GetAperture . . . 292
3.61.3.3 GetFocalLength . . . 292
3.61.3.4 GetExposure . . . 293
3.61.3.5 GetGain . . . 293
3.61.3.6 GetGamma . . . 293
3.61.3.7 SetAperture . . . 293
3.61.3.8 SetFocalLength . . . 293
3.61.3.9 SetExposure . . . 293
3.61.3.10 SetGain . . . 293
3.61.3.11 SetGamma . . . 294
3.62 HMetadataNorPixDMIExtended Class Reference . . . 294
3.62.1 Detailed Description . . . 294
3.62.2 Constructor & Destructor Documentation . . . 294
3.62.2.1 HMetadataNorPixDMIExtended . . . 294
3.62.2.2 HMetadataNorPixDMIExtended . . . 295
3.62.3 Member Function Documentation . . . 295
3.62.3.1 BuildMetadataString . . . 295
3.63 HMetadataNorPixDMIFBExtended Class Reference . . . 295
3.63.1 Detailed Description . . . 296
3.63.2 Constructor & Destructor Documentation . . . 296
3.63.2.1 HMetadataNorPixDMIFBExtended . . . 296
3.63.2.2 HMetadataNorPixDMIFBExtended . . . 296
3.63.3 Member Function Documentation . . . 296
3.63.3.1 BuildMetadataString . . . 296
3.64 HMetadataNorPixGPS Class Reference . . . 296
3.64.1 Detailed Description . . . 297
3.64.2 Constructor & Destructor Documentation . . . 297
3.64.2.1 HMetadataNorPixGPS . . . 297
3.64.2.2 HMetadataNorPixGPS . . . 297
3.64.3 Member Function Documentation . . . 297
3.64.3.1 BuildMetadataString . . . 297

3.65 HMetadataNorPixGPSExtended Class Reference . . . 297
3.65.1 Detailed Description . . . 298
3.65.2 Constructor & Destructor Documentation . . . 299
3.65.2.1 HMetadataNorPixGPSExtended . . . 299
3.65.2.2 HMetadataNorPixGPSExtended . . . 300
3.65.2.3 HMetadataNorPixGPSExtended . . . 300
3.65.2.4 HMetadataNorPixGPSExtended . . . 300
3.65.2.5 HMetadataNorPixGPSExtended . . . 301
3.65.3 Member Function Documentation . . . 301
3.65.3.1 BuildMetadataString . . . 301
3.66 HMetadataNorPixGRTHeader Class Reference . . . 301
3.66.1 Detailed Description . . . 301
3.66.2 Constructor & Destructor Documentation . . . 302
3.66.2.1 HMetadataNorPixGRTHeader . . . 302
3.66.2.2 HMetadataNorPixGRTHeader . . . 303
3.66.3 Member Function Documentation . . . 303
3.66.3.1 BuildMetadataString . . . 303
3.67 HMetadataNorPixImageStatistics Class Reference . . . 303
3.67.1 Detailed Description . . . 303
3.67.2 Constructor & Destructor Documentation . . . 304
3.67.2.1 HMetadataNorPixImageStatistics . . . 304
3.67.2.2 HMetadataNorPixImageStatistics . . . 305
3.67.3 Member Function Documentation . . . 305
3.67.3.1 BuildMetadataString . . . 305
3.68 HMetadataNorPixInputLineState Class Reference . . . 305
3.68.1 Detailed Description . . . 306
3.68.2 Constructor & Destructor Documentation . . . 306
3.68.2.1 HMetadataNorPixInputLineState . . . 306
3.68.2.2 HMetadataNorPixInputLineState . . . 306
3.68.3 Member Function Documentation . . . 306
3.68.3.1 BuildMetadataString . . . 306
3.68.3.2 GetDeviceName . . . 307
3.68.3.3 GetLineName . . . 307
3.68.3.4 GetInputLevel . . . 307
3.69 HMetadataNorPixLidar Class Reference . . . 307
3.69.1 Detailed Description . . . 308
3.69.2 Member Function Documentation . . . 308
3.69.2.1 BuildMetadataString . . . 308
3.70 HMetadataNorPixLTC Class Reference . . . 308
3.70.1 Detailed Description . . . 308

3.70.2 Constructor & Destructor Documentation . . . 308
3.70.2.1 HMetadataNorPixLTC . . . 308
3.70.2.2 HMetadataNorPixLTC . . . 309
3.70.3 Member Function Documentation . . . 309
3.70.3.1 BuildMetadataString . . . 309
3.70.3.2 GetLTC . . . 309
3.71 HMetadataNorPixTime64 Class Reference . . . 309
3.71.1 Detailed Description . . . 310
3.71.2 Constructor & Destructor Documentation . . . 310
3.71.2.1 HMetadataNorPixTime64 . . . 310
3.71.2.2 HMetadataNorPixTime64 . . . 310
3.71.3 Member Function Documentation . . . 310
3.71.3.1 BuildMetadataString . . . 310
3.72 HMotionDetection Class Reference . . . 310
3.72.1 Detailed Description . . . 312
3.72.2 Member Function Documentation . . . 312
3.72.2.1 SetErrorOutput . . . 312
3.72.2.2 GetLastError . . . 312
3.72.2.3 SetMotionDetectionMode . . . 312
3.72.2.4 GetMotionDetectionMode . . . 313
3.72.2.5 SetSobelDetectionMode . . . 313
3.72.2.6 GetSobelDetectionMode . . . 313
3.72.2.7 SetThresholdValue . . . 313
3.72.2.8 GetThresholdValue . . . 313
3.72.2.9 SetAbsoluteThresholdValue . . . 313
3.72.2.10 GetAbsoluteThresholdValue . . . 313
3.72.2.11 SetRoi . . . 314
3.72.2.12 GetRoi . . . 315
3.72.2.13 ConvertToGray . . . 315
3.72.2.14 MeanGrayDetect . . . 315
3.72.2.15 AbsDiffDetect . . . 315
3.72.2.16 AbsMaxDiffDetect . . . 317
3.72.2.17 EdgeDetect . . . 317
3.72.2.18 EdgeDetect . . . 317
3.72.2.19 AdaptiveEdgeDetect . . . 318
3.72.2.20 AdaptiveEdgeDetect . . . 318
3.72.2.21 OverlayImage . . . 318
3.72.2.22 MoveTowards . . . 319
3.72.2.23 SetErrorOutput . . . 319
3.72.2.24 GetLastError . . . 319

3.72.2.25 SetMotionDetectionMode . . . 319
3.72.2.26 GetMotionDetectionMode . . . 319
3.72.2.27 SetSobelDetectionMode . . . 319
3.72.2.28 GetSobelDetectionMode . . . 320
3.72.2.29 SetThresholdValue . . . 320
3.72.2.30 GetThresholdValue . . . 320
3.72.2.31 SetAbsoluteThresholdValue . . . 320
3.72.2.32 GetAbsoluteThresholdValue . . . 320
3.72.2.33 SetRoi . . . 320
3.72.2.34 GetRoi . . . 321
3.72.2.35 ConvertToGray . . . 321
3.72.2.36 MeanGrayDetect . . . 321
3.72.2.37 AbsDiffDetect . . . 321
3.72.2.38 AbsMaxDiffDetect . . . 322
3.72.2.39 EdgeDetect . . . 322
3.72.2.40 EdgeDetect . . . 322
3.72.2.41 AdaptiveEdgeDetect . . . 323
3.72.2.42 AdaptiveEdgeDetect . . . 323
3.72.2.43 OverlayImage . . . 323
3.72.2.44 MoveTowards . . . 324
3.73 HMP3File Class Reference . . . 324
3.73.1 Detailed Description . . . 325
3.73.2 Member Function Documentation . . . 325
3.73.2.1 SetErrorOutput . . . 325
3.73.2.2 GetLastError . . . 325
3.73.2.3 Create . . . 325
3.73.2.4 SetAudioParam . . . 326
3.73.2.5 AddData . . . 326
3.73.2.6 EndRecording . . . 326
3.73.2.7 Open . . . 326
3.73.2.8 GetAudioInfo . . . 327
3.73.2.9 GetTotalAudioDatas . . . 327
3.73.2.10 ReadAudio . . . 327
3.73.2.11 GetTimeStamp . . . 327
3.73.2.12 GetPositionByTimeStamp . . . 328
3.73.2.13 SetReadingPosition . . . 328
3.73.2.14 GetReadingPosition . . . 328
3.73.2.15 Close . . . 328
3.74 HMP4File Class Reference . . . 328
3.74.1 Detailed Description . . . 330

3.74.2 Member Function Documentation . . . 330
3.74.2.1 SetErrorOutput . . . 330
3.74.2.2 GetLastError . . . 330
3.74.2.3 Create . . . 331
3.74.2.4 SetVideoParam . . . 331
3.74.2.5 SetAudioParam . . . 331
3.74.2.6 PrepareWrite . . . 332
3.74.2.7 IsSuggestPushVideo . . . 332
3.74.2.8 AddFrame . . . 332
3.74.2.9 IsSuggestPushAudio . . . 332
3.74.2.10 AddAudio . . . 332
3.74.2.11 FinalizeWrite . . . 333
3.74.2.12 EndRecording . . . 333
3.74.2.13 Close . . . 333
3.74.2.14 SetCallbacks . . . 333
3.74.2.15 Open . . . 333
3.74.2.16 GetVideoInfo . . . 334
3.74.2.17 GetAudioInfo . . . 334
3.74.2.18 Read . . . 334
3.74.2.19 SetPlaybackPosition . . . 335
3.74.2.20 GetIndexAt . . . 335
3.74.2.21 GetActionMode . . . 335
3.74.2.22 Read . . . 335
3.74.2.23 GetAllocatedFrames . . . 336
3.74.2.24 GetTotalAudioDatas . . . 336
3.74.2.25 SetPlaybackFrameRate . . . 336
3.74.2.26 GetKeyFrameByIndex . . . 336
3.74.2.27 Play . . . 336
3.74.2.28 PlayFixedRate . . . 336
3.74.2.29 Stop . . . 337
3.74.2.30 IsPlaying . . . 337
3.74.2.31 ReadAudio . . . 337
3.75 HMPEGTsFile Class Reference . . . 337
3.75.1 Detailed Description . . . 338
3.75.2 Member Function Documentation . . . 338
3.75.2.1 SetErrorOutput . . . 338
3.75.2.2 GetLastError . . . 338
3.75.2.3 Create . . . 338
3.75.2.4 SetVideoParam . . . 339
3.75.2.5 SetAudioParam . . . 339

3.75.2.6 PrepareWrite . . . 339
3.75.2.7 AddFrame . . . 340
3.75.2.8 AddAudio . . . 340
3.75.2.9 FinalizeWrite . . . 340
3.75.2.10 EndRecording . . . 340
3.75.2.11 Close . . . 340
3.76 HRotation Class Reference . . . 341
3.76.1 Detailed Description . . . 342
3.76.2 Member Function Documentation . . . 342
3.76.2.1 SetErrorOutput . . . 342
3.76.2.2 GetLastError . . . 342
3.76.2.3 SetDestinationImageSize . . . 342
3.76.2.4 GetDestinationImageSize . . . 343
3.76.2.5 SetCenterCoordinate . . . 343
3.76.2.6 GetCenterCoordinate . . . 343
3.76.2.7 SetRotateAngle . . . 343
3.76.2.8 GetRotateAngle . . . 343
3.76.2.9 SetInterpolateMode . . . 344
3.76.2.10 GetInterpolateMode . . . 345
3.76.2.11 SetSmoothEdge . . . 345
3.76.2.12 GetSmoothEdge . . . 345
3.76.2.13 IsImageFormatSupported . . . 345
3.76.2.14 Rotate . . . 345
3.76.2.15 SetErrorOutput . . . 346
3.76.2.16 GetLastError . . . 346
3.76.2.17 SetDestinationImageSize . . . 346
3.76.2.18 GetDestinationImageSize . . . 346
3.76.2.19 SetCenterCoordinate . . . 346
3.76.2.20 GetCenterCoordinate . . . 346
3.76.2.21 SetRotateAngle . . . 347
3.76.2.22 GetRotateAngle . . . 347
3.76.2.23 SetInterpolateMode . . . 347
3.76.2.24 GetInterpolateMode . . . 347
3.76.2.25 SetSmoothEdge . . . 347
3.76.2.26 GetSmoothEdge . . . 347
3.76.2.27 IsImageFormatSupported . . . 348
3.76.2.28 Rotate . . . 348
3.77 HSequence Class Reference . . . 348
3.77.1 Detailed Description . . . 352
3.77.2 Member Function Documentation . . . 353

3.77.2.1 SetErrorOutput . . . 353
3.77.2.2 GetLastError . . . 353
3.77.2.3 SetCallbacks . . . 353
3.77.2.4 Create . . . 353
3.77.2.5 Open . . . 354
3.77.2.6 Close . . . 354
3.77.2.7 IsSequenceOpen . . . 354
3.77.2.8 IsRAMSequence . . . 354
3.77.2.9 Write . . . 355
3.77.2.10 Read . . . 355
3.77.2.11 SetRecordingPosition . . . 355
3.77.2.12 GetRecordingPosition . . . 356
3.77.2.13 SupportRewrite . . . 356
3.77.2.14 DumpHeader . . . 356
3.77.2.15 GetFrameRate . . . 356
3.77.2.16 SetFrameRate . . . 357
3.77.2.17 GetAllocatedFrames . . . 358
3.77.2.18 ReadTimestamp . . . 358
3.77.2.19 GetIndexAt . . . 358
3.77.2.20 GetIndexNear . . . 358
3.77.2.21 PreAllocateImages . . . 359
3.77.2.22 GetPreAllocatedImages . . . 359
3.77.2.23 EnableMetadata . . . 359
3.77.2.24 SetPlaybackPosition . . . 359
3.77.2.25 GetPlaybackPosition . . . 360
3.77.2.26 Play . . . 360
3.77.2.27 PlayFixedRate . . . 360
3.77.2.28 IsPlayFixedRate . . . 360
3.77.2.29 Stop . . . 361
3.77.2.30 SetPlaybackSpeed . . . 361
3.77.2.31 GetPlaybackSpeed . . . 361
3.77.2.32 IsPlaybackLagging . . . 361
3.77.2.33 SetPlaybackFrameSkip . . . 361
3.77.2.34 GetPlaybackFrameSkip . . . 362
3.77.2.35 SetMaxPlaybackInterval . . . 362
3.77.2.36 GetMaxPlaybackInterval . . . 362
3.77.2.37 IsPlaying . . . 362
3.77.2.38 SetPlaybackRange . . . 363
3.77.2.39 GetPlaybackRange . . . 363
3.77.2.40 ResynchronizePlayback . . . 363

3.77.2.41 ResynchronizePlayback . . . 363
3.77.2.42 GetNextPlaybackFrameTime . . . 364
3.77.2.43 GetImageWidth . . . 364
3.77.2.44 GetImageHeight . . . 364
3.77.2.45 GetImageBitDepth . . . 364
3.77.2.46 GetImageBitDepthReal . . . 364
3.77.2.47 GetImageSizeBytes . . . 365
3.77.2.48 GetImageFormat . . . 365
3.77.2.49 GetTrueImageSize . . . 365
3.77.2.50 GetSequenceCompression . . . 365
3.77.2.51 SetVideoSequenceParameter . . . 365
3.77.2.52 SetVideoEncoderType . . . 365
3.77.2.53 SetCompressionBayerPatternType . . . 366
3.77.2.54 GetCompressionBayerPattern . . . 366
3.77.2.55 SetCayerCompressionType . . . 366
3.77.2.56 GetCayerCompressionType . . . 366
3.77.2.57 SetCompressSeqIndexType . . . 366
3.77.2.58 GetCompressSeqIndexType . . . 367
3.77.2.59 SetAutoRecoveryIndexStream . . . 367
3.77.2.60 GetAutoRecoveryIndexStream . . . 367
3.77.2.61 RecoverIndexStream . . . 367
3.77.2.62 GetDescription . . . 367
3.77.2.63 SetDescription . . . 368
3.77.2.64 GetDescriptionFormat . . . 368
3.77.2.65 SetDescriptionFormat . . . 368
3.77.2.66 IsBayer . . . 368
3.77.2.67 SetBayerPattern . . . 368
3.77.2.68 GetBayerPattern . . . 369
3.77.2.69 SetReferenceFrame . . . 369
3.77.2.70 GetReferenceFrame . . . 369
3.77.2.71 HasReferenceTime . . . 369
3.77.2.72 SetReferenceTime . . . 369
3.77.2.73 GetReferenceTime . . . 370
3.77.2.74 SetJpegEncoderType . . . 370
3.77.2.75 GetJpegEncodeType . . . 370
3.77.2.76 SetJpegDecoderType . . . 370
3.77.2.77 GetJpegDecodeType . . . 370
3.77.2.78 EnableGPUJpegCompressionBayerInterp . . . 371
3.77.2.79 EnableGPUJpegCompressionColorprcessingLUT . . . 371
3.77.2.80 UpdateGPUJpegCompressionLUT . . . 371

3.77.2.81 Sort . . . 371
3.77.2.82 Truncate . . . 371
3.77.2.83 HasMarkers . . . 371
3.77.2.84 HasLTC . . . 372
3.77.2.85 SetTimeOffset . . . 372
3.77.2.86 SetOldestFrameIndex . . . 372
3.77.2.87 GetOldestFrameIndex . . . 372
3.78 HSequencerCallbacks Class Reference . . . 372
3.78.1 Detailed Description . . . 373
3.78.2 Member Function Documentation . . . 373
3.78.2.1 OnImageWrite . . . 373
3.78.2.2 OnImageRead . . . 373
3.78.2.3 OnQueryContinuePlayback . . . 373
3.78.2.4 OnEndOfSequence . . . 373
3.78.2.5 OnEndOfPlaybackRange . . . 374
3.78.2.6 OnPlaybackStopped . . . 374
3.79 HStreamDeliver Class Reference . . . 374
3.79.1 Detailed Description . . . 374
3.79.2 Member Function Documentation . . . 375
3.79.2.1 DeliverNewFrame . . . 375
3.79.2.2 DeliverNewAudio . . . 375
3.79.2.3 AddVirtualURL . . . 375
3.79.2.4 RemoveVirtualURL . . . 375
3.80 HStreamServer Class Reference . . . 376
3.80.1 Detailed Description . . . 376
3.80.2 Member Function Documentation . . . 377
3.80.2.1 RegisterSourceUrl . . . 377
3.80.2.2 UnRegisterSourceUrl . . . 377
3.80.2.3 SetHttpPort . . . 377
3.80.2.4 SetRtspPort . . . 377
3.80.2.5 SetRtspCredential . . . 378
3.80.2.6 StartServer . . . 378
3.80.2.7 SetCallback . . . 378
3.81 HTcpCallbacks Class Reference . . . 378
3.81.1 Detailed Description . . . 379
3.81.2 Member Function Documentation . . . 379
3.81.2.1 OnClientConnected . . . 379
3.81.2.2 OnReceiveData . . . 379
3.81.2.3 OnClientDisconnected . . . 379
3.82 HTcpClient Class Reference . . . 379

3.82.1 Detailed Description . . . 380
3.82.2 Member Function Documentation . . . 380
3.82.2.1 SetErrorOutput . . . 380
3.82.2.2 GetLastError . . . 380
3.82.2.3 SetCallback . . . 380
3.82.2.4 ConnectTo . . . 381
3.82.2.5 ConnectTo . . . 381
3.82.2.6 IsConnected . . . 381
3.82.2.7 Disconnect . . . 381
3.82.2.8 GetSocket . . . 382
3.82.2.9 Send . . . 382
3.82.2.10 GetPeerAddress . . . 382
3.82.2.11 GetPeerAddress . . . 382
3.83 HTcpServer Class Reference . . . 383
3.83.1 Detailed Description . . . 384
3.83.2 Member Function Documentation . . . 384
3.83.2.1 SetErrorOutput . . . 384
3.83.2.2 GetLastError . . . 384
3.83.2.3 SetCallback . . . 384
3.83.2.4 OpenServer . . . 384
3.83.2.5 OpenServer . . . 385
3.83.2.6 CloseServer . . . 385
3.83.2.7 IsServerOpen . . . 385
3.83.2.8 CloseConnection . . . 385
3.83.2.9 GetConnectionCount . . . 386
3.83.2.10 GetAddress . . . 386
3.83.2.11 GetAddress . . . 386
3.83.2.12 GetSocket . . . 386
3.83.2.13 GetFirstSocket . . . 387
3.83.2.14 GetNextSocket . . . 387
3.83.2.15 Send . . . 387
3.83.2.16 Broadcast . . . 387
3.83.2.17 GetPeerAddress . . . 388
3.83.2.18 GetPeerAddress . . . 388
3.84 Http_Request Struct Reference . . . 388
3.84.1 Detailed Description . . . 389
3.85 HTTPRequest_Callback Class Reference . . . 389
3.85.1 Detailed Description . . . 389
3.85.2 Member Function Documentation . . . 389
3.85.2.1 NewHttpRequest . . . 389

3.86 HUdpCallbacks Class Reference . . . 390
3.86.1 Detailed Description . . . 390
3.86.2 Member Function Documentation . . . 390
3.86.2.1 OnReceiveData . . . 390
3.87 HUdpClient Class Reference . . . 390
3.87.1 Detailed Description . . . 391
3.87.2 Member Function Documentation . . . 391
3.87.2.1 SetErrorOutput . . . 391
3.87.2.2 GetLastError . . . 391
3.87.2.3 SetCallback . . . 391
3.87.2.4 IsConnected . . . 391
3.87.2.5 Connect . . . 391
3.87.2.6 SendTo . . . 392
3.87.2.7 SendToEx . . . 392
3.88 HUdpServer Class Reference . . . 392
3.88.1 Detailed Description . . . 393
3.88.2 Member Function Documentation . . . 393
3.88.2.1 SetErrorOutput . . . 393
3.88.2.2 GetLastError . . . 393
3.88.2.3 SetCallback . . . 393
3.88.2.4 IsOpened . . . 394
3.88.2.5 Open . . . 394
3.88.2.6 GetIPAddress . . . 394
3.88.2.7 SendTo . . . 394
3.88.2.8 SendToEx . . . 395
3.89 HVideoOutputDriver Class Reference . . . 395
3.89.1 Detailed Description . . . 395
3.89.2 Member Function Documentation . . . 396
3.89.2.1 IsVideoOutputAvailable . . . 396
3.89.2.2 EnableVideoOutput . . . 396
3.89.2.3 DisableVideoOutput . . . 396
3.89.2.4 SendImageForVideoOutput . . . 396
3.90 HViewer Class Reference . . . 396
3.90.1 Detailed Description . . . 397
3.90.2 Member Function Documentation . . . 397
3.90.2.1 SetErrorOutput . . . 397
3.90.2.2 GetLastError . . . 397
3.90.2.3 SetZoom . . . 397
3.90.2.4 GetZoom . . . 398
3.90.2.5 ShowImage . . . 398

3.90.2.6 ShowScaledImage . . . 398
3.90.2.7 ShowImageOnDC . . . 399
3.91 HWhiteBalance Class Reference . . . 399
3.91.1 Detailed Description . . . 400
3.91.2 Member Function Documentation . . . 400
3.91.2.1 SetErrorOutput . . . 400
3.91.2.2 GetLastError . . . 400
3.91.2.3 GetColorBalance . . . 400
3.91.2.4 SetColorBalance . . . 400
3.91.2.5 GetAlgorithm . . . 401
3.91.2.6 SetAlgorithm . . . 401
3.91.2.7 GetAutoColorBalanceMode . . . 401
3.91.2.8 SetAutoColorBalanceMode . . . 401
3.91.2.9 GetColorBalanceApplied . . . 401
3.91.2.10 SetColorBalanceApplied . . . 401
3.91.2.11 ComputeColorBalance . . . 401
3.91.2.12 ComputeColorBalance . . . 403
3.91.2.13 ApplyColorBalance . . . 403
3.91.2.14 ApplyColorBalance . . . 403
3.91.2.15 ApplyColorBalance . . . 403
3.92 Http_Request::mg_header Struct Reference . . . 404

4 File Documentation **405**
4.1 HEnums.h File Reference . . . 405
4.1.1 Enumeration Type Documentation . . . 413
4.1.1.1 eHSupportedModule . . . 413
4.1.1.2 eHGrabberInfoType . . . 417
4.1.1.3 eHJpegQuality . . . 417
4.1.1.4 eHImageFormat . . . 417
4.1.1.5 eHCompression . . . 418
4.1.1.6 eHDescriptionFormat . . . 419
4.1.1.7 eHProtectionStatus . . . 419
4.1.1.8 HTimeSourceSyncStatus . . . 419
4.1.1.9 eHBayerSampleFactor . . . 419
4.1.1.10 eHBayerPatternOrigin . . . 419
4.1.1.11 eHInputEvent . . . 420
4.1.1.12 eHInputLevel . . . 420
4.1.1.13 eHOutputLevel . . . 420
4.1.1.14 eHSpecialCopyFormat . . . 420
4.1.1.15 eAdvanceSettingsFeatureType . . . 421

4.1.1.16 eFastColorCorrectionMode . . . 421
4.1.1.17 eHImageDateTimeFormat . . . 421
4.1.1.18 eHMovieMode . . . 421
4.1.1.19 eHAviPlaybackFormat . . . 422
4.1.1.20 eHRotationInterpolationMode . . . 422
4.1.1.21 eHResizeInterpolationMode . . . 422
4.1.1.22 eHMotionDetectionMode . . . 422
4.1.1.23 eHSobelMode . . . 422
4.1.1.24 eMonoToPseudoColorLUT . . . 423
4.1.1.25 eHGRABBER_NOTIFICATION . . . 423
4.1.1.26 eHSUGGESTED_ACTION . . . 423
4.1.1.27 eHMirror_Mode . . . 424
4.1.1.28 eHMerge_Mode . . . 424
4.1.1.29 eAudioFileFormat . . . 424
4.1.1.30 eAudFileMode . . . 424
4.1.1.31 eHAudioFormat . . . 424
4.1.1.32 eHAudioCompression . . . 425
4.1.1.33 eHAudioPlayEventCode . . . 425
4.1.1.34 eHExportFormat . . . 425
4.1.1.35 eDAQDeviceParamType . . . 425
4.1.1.36 eTimeSourceParamType . . . 426
4.1.1.37 eHCayer_Compression . . . 426
4.1.1.38 eHMetadataFormat . . . 426
4.1.1.39 eDeviceAcclType . . . 426
4.1.1.40 eJpegEncodeAlgoType . . . 427
4.1.1.41 eJpegImageExportType . . . 427
4.1.1.42 eJpegDecodeAlgoType . . . 427
4.1.1.43 eHErrorCode . . . 427
4.2 HermesUtils.h File Reference . . . 431
4.3 MetadataID.h File Reference . . . 432
4.3.1 Detailed Description . . . 433

Index **435**

Chapter 1

Hermes API

1.1 Introduction

Hermes supports Windows Vista, 7 and 8. Previous versions (i.e. Windows 3.1/NT/95/98/ME/2000/XP) are not supported. The Hermes library is now compiled with VC++ 2012 and uses WinSxS.

To use the Hermes API, include the "hermes.h" and link it with the "hermesAPI.lib" library. The HermesAPI.dll must be copied in the same folder as your executable. It could also be placed in the windows/system folder, but this is not recommended. The HermesAPI.lib is built under Visual Studio 2012 and thus may or may not work with a previous version of Visual Studio.

Please contact `sales@norpix.com` to get authorization codes, otherwise most Hermes API classes can't be instantiated.

1.2 Definitions

Here are some basic definitions of concepts used within the Hermes API and its documentation:

Grabber: Used to designate a camera, frame grabber or any such image capturing device.

Image width : The width, in pixels, of an image.

Image height : The height, in pixels, of an image.

Image bitdepth : The number of bits used to store a single pixel. For example, the value for monochrome images (256 shades) will be 8 bits. Standard BGR images will be 24 bits (8 bits x 3 channels).

Real image bitdepth : The number of bits holding relevant information on each channel. For example, a 16 bits monochrome camera often use 10 or 12 bits to store information. In that case, the Image BitDepth would be 16 and the real image bitdepth would be 10 or 12. In the case of BGR 48 bits image, each pixel is stored in 48 bits, allocating 16 bits per color.

Image format : This is the format in which the image information is stored. Ex : Monochrome, BGR, RGB, YUV422, etc.

1.3 Files included in this distribution

- bin/freeimage.dll
- bin/libtiff.dll
- bin/libjpeg.dll
- bin/zlib.dll
- bin/ijl20.dll
- bin/FastAlgo.dll
- bin/cfitsio.dll
- bin/IppWrapper.dll
- bin/EuresysJPEG.dll (1)
- bin/EasyMs60.dll (1)
- bin/ETools.dll (1)

Place these in the same folder as the application's executable. These libraries are used by the image exporters and sequence files and need to be bundled with your application.

(1) = You only need these if you want to support the Euresys frame grabbers.

Program Files/Common Files/Norpix/devices/
This folder contains the grabbers dll. They must be registered before use and need to be bundled with your application. Some grabbers have their own subfolder since they use manufacturer dlls which can not be downloaded or obtained normally.

Program Files/Common Files/Norpix/NpxGrabMan.dll
For the application to work, this file must be registered and needs to be bundled with your application.

Program Files/Common Files/Norpix/NpxGrabprop.dll
This file must be registered to access the property pages of any grabber and needs to be bundled with your application.

Program Files/Common Files/Norpix/ImgFeederFilter.ax
This file must be registered to enable AVI exporting and needs to be bundled with your application.

bin/NpxDebug.exe
This little program allows to trap debug messages sent to the system debugger, enabling display of the messages from drivers & Info Strings.

bin/SysInfo.exe
This program allows to easily retrieve the information needed to be granted authorization codes. (MAC address or HDD number) This also allows benchmarking of the hard disk drives speed.

bin/dll register.reg
When double-clicked on it, this optional file will add special instruction to the registry, allowing to simply double-click on dll files to register them. (This simply associate regsvr32 as the default program used when double-clicking a dll file.)

bin/vcredist_x86 SP1.exe
Microsoft installer to install all the required WinSxS DLLs on client computers.

doc/Hermes.pdf
The Hermes API documentation in PDF format.

doc/Drivers.pdf
Information on 3rd party drivers that need to be installed for the HermesAPI drivers to successfully register themselves.

Demo Projects
Every dll from the "bin" folder needs to be copied to the demo's "Release" or "Debug" folder.

- DemoGrab : A demo project that uses the "grabber", "sequencer" et "viewer" modules.
- DemoPlay : A demo project that uses the "sequencer" and "viewers" modules.
- DemoSeqToImg : A demo project that uses "sequencer" and "image exporter" modules.
- DemoSeqToAvi : A demo project that uses "sequencer" and "avi exporter" modules.

1.4 Windows Platform SDK & Direct X

It is recommended to install the latest Windows Platform SDK available for download at : http://www.microsoft.com/msdownload/platformsdk/sdkupdate

Having the "DirectX 9.0 SDK Update" is also necessary. It can be downloaded at: http://msdn.microsoft.com/downloads/

1.5 Unicode Support

If the application is Unicode, it must be linked to the Unicode version of the API.

	LIBRARY	DLL
NON UNICODE	HermesAPI.lib	HermesAPI.dll
UNICODE	HermesAPIu.lib	HermesAPIu.dll

In unicode, each character is stored on 16 bits instead of 8 bits. For instance, this allows Unicode applications to support languages like Japanese which have more than 256 characters in their "alphabet". To add Unicode support to the application, replace the Preprocessor definition "_MBCS" by "UNICODE, _UNICODE" in the project settings. The proper Unicode MFC libraries mus also be installed from the Visual C++ CD-ROM.

IMPORTANT : If you use the unicode library with Visual C++ 2005 or greater : Open the project's Property Pages dialog box. In the C/C++ section, open the Language page and modify the "Treat wchar_t as Built-in Type" to "No". You will get linking errors with Hermes API if you don't set this properly.

For more information about Unicode, please check MSDN documentation (see TCHAR).

1.6 Hermes and multi-threading

All Hermes API classes functions can be called from different threads. However, it is recommended to use locks (mutex, semaphores or critical sections) to synchronize access from different threads. For example, you must not allow a thread to call HSequence::Close() while another thread is in HSequence::Write().

IMPORTANT : A large part of the API is built upon the COM architecture (every DLL that must be "registered" is a COM DLL). To allow the use of COM objects in its space, a thread must call CoInitialize(NULL) before using any COM objects and CoUninitialize() when has finished using them. Each CoInitialize must have a matching CoUninitialize.

Classes such as HGrabModule, HAVIExporter and HAviFile make extensive uses of COM objects (for grabbers & codecs). As those classes implicitly call CoInitialize(NULL) in their constructor, there is no need for the programmer to call it unless the class instance will be used from another thread. In such a case, CoInitialize should be the first function called upon thread creation.

1.7 Hermes in a Window NT Service

As the Hermes API was not designed to run as a Windows service, it should never be used as a part of such an application.

1.8 Registering drivers

Hermes grabber drivers are COM dll. To be usable, they have to be registered by the system.

An Hermes Driver dll can be registered in the command prompt by using the "regsvr32" command. For example, "regsvr32 NpxQimaging.dll". If registration is successful, the HGrabModule Load function could then be used to load the Hermes QImaging driver. The authorization code for QImaging along with compatible driver versions must also be at hand.

A dll registration failure could mean that it could not find one of the dll dependency. Missing files can be listed by using a dependency tool such as "Depends", which ships with Visual C++ 6.0 and greater. The QImaging camera, for instance, needs a Hermes called NpxQImaging.dll. To register correctly, it needs to have access to the QCA↩MDRIVER.dll, a dll that only installed trough the QCaptureSuite software (available from the manufacturer web site or on CD).

Each Hermes driver is meant to be used with a specific version of the grabber driver, usually the one provided by the grabber's manufacturer at the time of Hermes API release.

IMPORTANT : It is also necessary to register NpxGrabman.dll and NpxGrabprop.dll for API to connect to the drivers and display their property pages.

For the image exporter module to work correctly (HImageExporter), the following files need to be present in the same folder as the executable : freeimage.dll, libtiff.dll, libjpeg.dll, zlib.dll.

For the sequence module to work correctly (HSequence), the following files need to be present in the same folder as the executable : ijl20.dll.

If needed, register the ImageFeederFilter (ImgFeederFilter.ax) to enable the AVI Exporter module (HAviExporter).

1.9 Hermes API Protection

As most modules in the Hermes API are protected trying to use an API or driver for which the authorization are missing will return a "H_ERROR_NOAUTHORIZATION" error.

Please contact `sales@norpix.com` to obtain a registry file (.npx) in which the authorizations codes will be written.

To license HERMES, simply run the SysInfo.exe utility provided with your HERMES distribution in the fololowing folder: C:/Program Files/Norpix/Hermes/bin64 SysInfo.exe includes a button to load the .npx file.

Alternatively, The npx files are actually .reg files which you can simply rename to .reg files and double-click on them to write the authorization codes to the system registry so they can be found by the Hermes API runtimes.

4 authorization methods are available :

- **External USB Key**
  The major advantage of this kind of protection is that it allows to switch your codes from one system to another. In order for Hermes to read the license on the USB Key, a third party software driver must be installed on the computer. The USB Key needs to stay connected to the system's USB port while Hermes API runtimes are loaded. USB key installation: Please run "Sentinel Protection Installer.exe" Latest version can be download from http://www.norpix.com/Download/SentinelInstaller.exe
- **Machine Code**
  This is the favored authorization method to be used if the external USB key method is not suitable. The machine code is built from a number of system parameters and can be retrieved by using SysInfo.exe.
- **MAC Address**
  The authorization code for a MAC address protection will allow the system with a matching MAC address to run the specified Hermes runtimes. The MAC address is an unique identifier specific to each Ethernet network card. To retrieve it, either use Hermes API's "sysInfo.exe", type "ipconfig /all" in the command prompt (the MAC address is called "Physical Address") or use the Windows Platform SDK "GetAdaptersInfo" function to retrieve it.
- **HDD s/n**
  The last protection method uses the serial number of the system primary hard disk drive (the drive where Windows is installed). However, should a primary hard disk failure, reformat or replacement occur, a new authorization code will be required.

Developer Environment Setup Check List

- Download and install the latest Windows Platform SDK.
- Download and install DirectX 9.0 SDK Update (release).
- Follow the "User Environment Setup Check List" for the remaining steps.

1.10 User Environment Setup Check List

- Make sure the user system is Windows Vista or better.
- Make sure the client has DirectX 9.0 or greater installed. Older versions might work but this has not been tested.
- Install HermesAPI.dll (or HermesAPIu.dll for Unicode application) in the same folder as the application ".exe".
- Install and register NpxGrabman.dll and NpxGrabprop.dll.
- Install FastAlgo.dll & IppWrapper.dll in the same folder as the ".exe" application.
- Install and register the drivers dll supported by the application. Warning : As the driver dll will refuse to register itself if it is not able to find all dependencies, manufacturer dlls must be installed prior to this step. A list of every supported grabber and its dll's name is available in HEnums.h.
- Install "ijl20.dll", "freeimage.dll", "libtiff.dll", "libjpeg.dll" and "zlib.dll" in the same folder as the ".exe" application.

- Install the "vcredist_x86 SP1.exe".
- If implementing the HAviExporter, install and register "ImgFeederFilter.ax" on the system.
- If you use the .NET wrapper, install the latest .NET framework from WindowsUpdate.
- Contact NorPix sales (sales@norpix.com) to get authorization codes for the user system. The favored protection method is by "MAC Address", then comes the USB Key and in last resort , the "HDD s/n" method. "MAC Address" and "HDD s/n" can be retrieved by using Hermes API's "SysInfo.exe".

1.11 Organization

The Hermes API is a small suite of modules :

Grabber Module : Assigned to a frame grabber, it capture frames in HImage wrappers.

Image Exporter Module : Export HImages to BMP, JPEG, TIFF, PNG, JPEG2000, FIT.

AVI Exporter Module : Exports HImages to AVI files. (obsolete, use HAviFile for new projects)

AVI Recording & Playback Module : Manage the recording and playback of AVI files.

Sequence Module : Reads and writes HImage to and from a sequence file (.seq) on disk by using HsequencerDisk or to RAM by using HSequencerRAM.

Viewer Module : Displays a HImage on a specified Device Context.

FPS Monitor Module : Computes the frame rate of any specified events.

The modules also use the following support classes :

HImage : Includes the data of a captured image.

HItem : Represents a single item of the lists returned by the HAviExporter module.

1.12 About time_t

time_t is used to represent an absolute time. In Hermes, the time stamps are always handled with 3 values :

- a "long" time : the time elapsed since midnight (00:00:00), January 1, 1970 (in seconds).
- a "unsigned short" timeMS : the millisecond part.
- a "unsigned short" timeUS : the microsecond part.

Hermes uses the 32-bits version of time_t time variable. It is considered as a "long" time to work around the linking problems arising with Visual Studio 2005 and higher, in which time_t is a 64 bits structure. For more information on the time_t structure, please refer to the MSDN Library.

Chapter 2

Module Documentation

2.1 Acquisition

Everything to acquire data from hardware devices.

Modules

- Video

  Capture from the camera / frame grabber.
- Audio

  Capture from an audio device.
- Data (DAQ)

  Capture data.

2.1.1 Detailed Description

Everything to acquire data from hardware devices.

2.2 Video

Capture from the camera / frame grabber.

Classes

- class HFPSMonitor

  This class can be used to easily compute a frame rate.
- class HFPSMonitorCallbacks

  Allows the user to be notified when specific events occur in the HFPSMonitor.
- class HGrabModule

  This class manages image acquisition from a camera.
- class HGrabCallbacks

  Allows the user to be notified when specific events occur in the HGrabModule.
- class HImage

  The image container used everywhere in the API.

2.2.1 Detailed Description

Capture from the camera / frame grabber.

2.3 Audio

Capture from an audio device.

Classes

- class HAudioData

  An audio buffer holding a range of samples.
- class HAudioDriverASIO

  Manage an ASIO sound device.
- class HAudioDriverDirectX

  Manage a DirectX sound device.
- class HAudioRecorder

  Control audio recording from an audio device to an audio file.
- class HAudioPlayer

  This class can be use to control audio playback from an .AUD ,.MP4, MP3 or AAC audio file.

2.3.1 Detailed Description

Capture from an audio device.

2.4 Data (DAQ)

Capture data.

Classes

- class CDaqSupportCallback

  This class manages the DAQ devices callbacks.
- class HDaqFileCallbacks

  This class manages the DAQ playback's callbacks.
- class HDaqDataSet

  This class represent a single DAQ data set.
- class HDaqFile

  This class handle a DAQ file and all related operations.

2.4.1 Detailed Description

Capture data.

2.5 Image Processing

Image processing that can be applied on HImages.

Classes

- class HBitmapOverlay

  This module can used to overlay a bitmap to an image.
- class HFrameAverage

  Frame averaging.
- class HGrayConverter

  This class converts an HImage to grey monochrome 8 bit format.
- class HImageColorProcessing

  This class applies a Bayer conversion on a raw image and/or does a color balance to an image.
- class HImageConverter

  This class converts an HImage from one format to another.
- class HImageMerge

  Merge 2 images together to create a larger image.
- class HImageMirror

  Flip an image (horizontaly or verticaly).
- class HImageOverlay

  Write overlays on any supported image.
- class HImageResize

  Changes an image size using different interpolation algorithms.
- class HJpegCompression

  This class JPEG compress or decompress an HImage This class can be used to jpeg compress/decompress.
- class HLineScanManager

  This class is used for managing images originated from a line scan sensor.
- class HMotionDetection

  This is an utility class that allows to detect object motion in images.
- class HRotation

  This is an utility class that can be used to handle image rotation.
- class HWhiteBalance

  This class applies color balance for color image.

2.5.1 Detailed Description

Image processing that can be applied on HImages.

2.6 File Formats

Recording / Playback / Export.

Modules

- AVI File

  AVI file recording / playback / export.
- MOV File

  MOV file recording.
- SEQ File

  Sequence file recording / playback / export.
- Image Files

  Image files recording / export.

2.6.1 Detailed Description

Recording / Playback / Export.

2.7 AVI File

AVI file recording / playback / export.

Classes

- struct CCineFormParam

  The structure holding the the Cineform parameters.
- struct CCineFormRawParam

  Cineform RAW CODEC parameter.
- struct CCineFormHDParam

  Cineform HC CODEC parameter.
- class HAviExporter

  Export a sequence file to an AVI.
- class HAviFile

  This class manages AVI recording & playback.
- class HAviFileCallbacks

  This class handles the callbacks sent by the HAviFile playback functions.

Enumerations

- enum CineFormCurve {
  H_CURVE_GAMMA22 = 0, **H_CURVE_GAMMA18**, **H_CURVE_LOG90**, **H_CURVE_GAMMA_CUSTOM**, **H_CURVE_LOG_CUSTOM**, **H_CURVE_LINE_CUSTOM** }

  Cineform curve mode.

2.7.1 Detailed Description

AVI file recording / playback / export.

2.8 MOV File

MOV file recording.

MOV file recording.

2.9 SEQ File

Sequence file recording / playback / export.

Classes

- class HSequence

  This class manages sequence files for recording and playback.
- class HSequencerCallbacks

  Allows the user to be notified when specific events occur in the HSequence.

2.9.1 Detailed Description

Sequence file recording / playback / export.

2.10 Image Files

Image files recording / export.

Classes

- class HImageExporter

  This class takes one or more images in HImage format and saves them in stand-alone files on disk.
- class HItem

  This is a utility class used by some other classes.

2.10.1 Detailed Description

Image files recording / export.

2.11 I/O devices

Manage Input / Output devices.

Classes

- class HIODriver

  Manage an I/O port.

Functions

- typedef void (HERMESCALL ∗InputSupportCallback)(int deviceIndex

  The input support callback.

2.11.1 Detailed Description

Manage Input / Output devices.

2.12 Utilities

Various utilities.

Classes

- class HErrorManager

  This class translate Hermes error codes to text strings.
- class HLocker

  A mutex class used through the API.

Functions

- HERMES_API LPCTSTR HImageFormatString (eHImageFormat format)

  Retreive the string describing an image format ID.
- HERMES_API LPCTSTR HImageCompressionString (eHCompression compression)

  Retreive the string describing a compression format ID.
- HERMES_API bool IsFormatMono (eHImageFormat format)

  Check if a given image format is a kind of monochrome image.
- HERMES_API bool IsFormatToF (eHImageFormat format)

  Check if a given image format is a kind of ToF image.
- HERMES_API void AbsoluteToRelative (long reference, unsigned short referenceMS, unsigned short referenceUS, long absolute, unsigned short absoluteMS, unsigned short absoluteUS, long &relative, unsigned short &relativeMS, unsigned short &relativeUS)

  Convert an absolute time to a relative time given a reference time.
- HERMES_API void RelativeToAbsolute (long reference, unsigned short referenceMS, unsigned short referenceUS, long relative, unsigned short relativeMS, unsigned short relativeUS, long &absolute, unsigned short &absoluteMS, unsigned short &absoluteUS)

  Convert a relative time to an absolute time a reference time.
- HERMES_API bool ComputeTimeDifference (long time1, unsigned short time1MS, unsigned short time1U↩S, long time2, unsigned short time2MS, unsigned short time2US, long &diffTime, short &diffTimeMS, short &diffTimeUS, bool &time2IsGreater)

  Compute the time delta between two absolute times.
- HERMES_API int CompareTimestamps (long time1, unsigned short time1MS, unsigned short time1US, long time2, unsigned short time2MS, unsigned short time2US)

  Compare two times to check which one is the greater.
- HERMES_API void AddTimestampOffset (long &time, short &timeMS, short &timeUS, long offset, short offsetMS, short offsetUS)

  Add a time offset to an existing time value.
- HERMES_API void CreateDirectoryRecursive (LPCTSTR filepath)

  Make sure every folders in the given path exist and create them otherwise.
- HERMES_API double ReadAVIFrameRate (LPCTSTR filePath)

  Read the frame rate value from an AVI file.
- HERMES_API void WriteAVIFrameRate (LPCTSTR filePath, double framerate)

  Overwrite the frame rate of an AVI file.
- int HGPUDeviceManager::GetNvidiaGPUDeviceCount ()

  Get NVIDIA GPU count.
- bool HGPUDeviceManager::IsNvidiaH264Present (int device)

  Check if NVIDIA driver supports GPU H.264 encoding.
- bool HGPUDeviceManager::IsNvidiaGPUHEVCPresent (int device)

  Check if NVIDIA driver supports GPU HEVC encoding.

- int HGPUDeviceManager::GetAMDGPUDeviceCount ()

  Get AMD GPU count.
- bool HGPUDeviceManager::IsAMDH264Present (int device)

  Check if AMD driver supports GPU H.264 encoding.

2.12.1 Detailed Description

Various utilities.

2.12.2 Function Documentation

2.12.2.1 HERMES_API LPCTSTR HImageFormatString (eHImageFormat *format*)

Retreive the string describing an image format ID.

Returns

The format description.

Parameters

in	*format*	The format ID.

2.12.2.2 HERMES_API LPCTSTR HImageCompressionString (eHCompression *compression*)

Retreive the string describing a compression format ID.

Returns

The format description.

Parameters

in	*compression*	The format ID.

2.12.2.3 HERMES_API bool IsFormatMono (eHImageFormat *format*)

Check if a given image format is a kind of monochrome image.

Returns

True if it is a monochrome format, false otherwise.

Parameters

in	*format*	The image format to check

2.12.2.4 HERMES_API bool IsFormatToF (eHImageFormat *format*)

Check if a given image format is a kind of ToF image.

Returns

True if it is a ToF format, false otherwise.

2.12.2.5 HERMES_API void AbsoluteToRelative (long *reference,* unsigned short *referenceMS,* unsigned short *referenceUS,* long *absolute,* unsigned short *absoluteMS,* unsigned short *absoluteUS,* long & *relative,* unsigned short & *relativeMS,* unsigned short & *relativeUS*)

Convert an absolute time to a relative time given a reference time.

Relative time = absolute time - reference time.

Parameters

in	*reference*	The seconds part of the reference time (32-bit time_t).
in	*referenceMS*	The milliseconds part of the reference time.
in	*referenceUS*	The microseconds part of the reference time.
in	*absolute*	The seconds part of the absolute time (32-bit time_t).
in	*absoluteMS*	The milliseconds part of the absolute time.
in	*absoluteUS*	The microseconds part of the absolute time.
out	*relative*	The seconds part of the relative time
out	*relativeMS*	The milliseconds part of the relative time.
out	*relativeUS*	The microseconds part of the relative time.

2.12.2.6 HERMES_API void RelativeToAbsolute (long *reference,* unsigned short *referenceMS,* unsigned short *referenceUS,* long *relative,* unsigned short *relativeMS,* unsigned short *relativeUS,* long & *absolute,* unsigned short & *absoluteMS,* unsigned short & *absoluteUS*)

Convert a relative time to an absolute time a reference time.

Absolute time = reference time + relative time.

Parameters

in	*reference*	The seconds part of the reference time (32-bit time_t).
in	*referenceMS*	The milliseconds part of the reference time.
in	*referenceUS*	The microseconds part of the reference time.
in	*relative*	The seconds part of the relative time
in	*relativeMS*	The milliseconds part of the relative time.
in	*relativeUS*	The microseconds part of the relative time.
out	*absolute*	The seconds part of the absolute time (32-bit time_t).
out	*absoluteMS*	The milliseconds part of the absolute time.
out	*absoluteUS*	The microseconds part of the absolute time.

2.12.2.7 HERMES_API bool ComputeTimeDifference (long *time1,* unsigned short *time1MS,* unsigned short *time1US,* long *time2,* unsigned short *time2MS,* unsigned short *time2US,* long & *diffTime,* short & *diffTimeMS,* short & *diffTimeUS,* bool & *time2IsGreater*)

Compute the time delta between two absolute times.

The time delta will always be a positive value, use the time2IsGreater parameter to check if the difference time is actually positive or negative.

Parameters

in	*time1*	The seconds part of the first time (32-bit time_t).
in	*time1MS*	The milliseconds part of the first time.
in	*time1US*	The microseconds part of the first time.
in	*time2*	The seconds part of the second time (32-bit time_t).

in	*time2MS*	The milliseconds part of the second time.
in	*time2US*	The microseconds part of the second time.
out	*diffTime*	The seconds part of the delta time.
out	*diffTimeMS*	The milliseconds part of the delta time.
out	*diffTimeUS*	The microseconds part of the delat time.
out	*time2IsGreater*	True if Time 2 is greater Time 1, false otherwise.

2.12.2.8 HERMES_API int CompareTimestamps (long *time1,* unsigned short *time1MS,* unsigned short *time1US,* long *time2,* unsigned short *time2MS,* unsigned short *time2US*)

Compare two times to check which one is the greater.

Returns

A negative value if time 1 $<$ time 2, a positive value if time 1 $>$ time 2, 0 if equal.

Parameters

in	*time1*	The seconds part of the first time (32-bit time_t).
in	*time1MS*	The milliseconds part of the first time.
in	*time1US*	The microseconds part of the first time.
in	*time2*	The seconds part of the second time (32-bit time_t).
in	*time2MS*	The milliseconds part of the second time.
in	*time2US*	The microseconds part of the second time.

2.12.2.9 HERMES_API void AddTimestampOffset (long & *time,* short & *timeMS,* short & *timeUS,* long *offset,* short *offsetMS,* short *offsetUS*)

Add a time offset to an existing time value.

Parameters

in,out	*time*	The seconds part of the absolute time (32-bit time_t).
in,out	*timeMS*	The milliseconds part of the absolute time.
in,out	*timeUS*	The microseconds part of the absolute time.
in	*offset*	How many seconds to add (or substract)
in	*offsetMS*	How many milliseconds to add (or substract)
in	*offsetUS*	How many microseconds to add (or substract)

2.12.2.10 HERMES_API void CreateDirectoryRecursive (LPCTSTR *filepath*)

Make sure every folders in the given path exist and create them otherwise.

Parameters

filepath	The filepath to check (or create)

2.12.2.11 HERMES_API double ReadAVIFrameRate (LPCTSTR *filePath*)

Read the frame rate value from an AVI file.

Returns

the AVI frame rate (in frames per second).

Parameters

in	*filePath*	Full file path to the AVI file.

2.12.2.12 HERMES_API void WriteAVIFrameRate (LPCTSTR *filePath,* double *framerate*)

Overwrite the frame rate of an AVI file.

Parameters

in	*filePath*	Full file path to the AVI file.
	framerate	The AVI frame rate (in frames per second).

2.12.2.13 bool HGPUDeviceManager::IsNvidiaH264Present (int *device*)

Check if NVIDIA driver supports GPU H.264 encoding.

Parameters

in	*device*	NVIDIA device index.

2.12.2.14 bool HGPUDeviceManager::IsNvidiaGPUHEVCPresent (int *device*)

Check if NVIDIA driver supports GPU HEVC encoding.

Parameters

in	*device*	NVIDIA device index.

2.12.2.15 bool HGPUDeviceManager::IsAMDH264Present (int *device*)

Check if AMD driver supports GPU H.264 encoding.

Parameters

in	*device*	AMD device index.

2.13 Metadata

Metadata handling.

Modules

- Generic Metadata

  Generic types of metadata.
- NorPix Metadata

  NorPix 1st-party metadata.

Files

- file MetadataID.h

  This file hold the unique identifiers for all generic metadata types along with all the metadata types created by NorPix.

Classes

- class HMetadata

  Base class that manages a metadata information.
- class HMetadataInfo

  Manage all the information needed to handle each metadata type.
- class HMetadataManager

  Manage the metadata information stored in the registry.

2.13.1 Detailed Description

Metadata handling.

A metadata is a piece of information that can be attached to an image. Accessing the metadatas attached to an image is done through the functions of the Image class. While the Metadata class represent the generic container for the metadata, the classes derived from Metadata are used to read/write and format the the actual data. If you want to use metadata, take some time to check the code in the various header files (.h) provided in Hermes.

2.14 Generic Metadata

Generic types of metadata.

Classes

- class HMetadataGenericBool

  1 byte "bool" value
- class HMetadataGenericByte

  1 byte value
- class HMetadataGenericShort

  2 bytes "short" value HMetadataGenericBinary - binary data (variable length)
- class HMetadataGenericUShort

  2 bytes "unsigned short" value
- class HMetadataGenericInt

  4 bytes "int" value
- class HMetadataGenericUInt

  4 bytes "unsigned int" value
- class HMetadataGenericInt64

  8 bytes "int64" value
- class HMetadataGenericUInt64

  8 bytes "unsigned int64" value
- class HMetadataGenericDouble

  8 bytes "double" value
- class HMetadataGenericString

  "wchar_t" null-terminated string (variable length)
- class HMetadataGenericBinary

  Binary data of variable length.

2.14.1 Detailed Description

Generic types of metadata.

Those classes all have a constructor with a customID parameter that allow 3rd party to use them to manage simple types as metadata.

2.15 NorPix Metadata

NorPix 1st-party metadata.

Classes

- class HMetadataNorPixTime64

  Timestamp in time_t format (64-bit) along with MS and US precision.
- class HMetadataNorPixLTC

  LTC value HH:MM:SS:FF.
- class HMetadataNorPixGPS

  GPS data (latitude, longitude)
- class HMetadataNorPixGPSExtended

  GPS extended data (latitude, longitude, altitude, speed, bearing, time)
- class HMetadataNorPixLidar

  Lidar data.
- class HMetadataNorPixGRTHeader

  GRT Header.
- class HMetadataNorPixDMIExtended

  8 bytes "DMI front mile post" value
- class HMetadataNorPixDMIFBExtended

  16 bytes "DMI front and back mile post" value
- class HMetadataNorPixInputLineState

  Input line state.
- class HMetadataNorPixCameraInfo

  Camera info.
- class HMetadataNorPixImageStatistics

  Image Statistics.

2.15.1 Detailed Description

NorPix 1st-party metadata.

Various part of the API will create those specific metadata types.

2.16 Network

Network management.

Modules

- UDP Network

  UDP connections management.
- TCP Network

  TCP connections management.

2.16.1 Detailed Description

Network management.

2.17 UDP Network

UDP connections management.

Classes

- class HUdpCallbacks

  Manage TCP Callbacks.
- class HUdpClient

  Manage a UDP client.
- class HUdpServer

  Manage a UDP server.

2.17.1 Detailed Description

UDP connections management.

2.18 TCP Network

TCP connections management.

Classes

- class HTcpCallbacks

  Manage TCP Callbacks.
- class HTcpClient

  Manage a TCP client.
- class HTcpServer

  Manage a TCP server.

2.18.1 Detailed Description

TCP connections management.

2.19 Viewer

To display HImage.

Classes

- class HVideoOutputDriver

  Control a video output port.
- class HViewer

  This class provides functions to draw HImages on-screen.

2.19.1 Detailed Description

To display HImage.

Chapter 3

Class Documentation

3.1 CAudioCaptureCallback Class Reference

This class is used by the HAudioDriverDirectX and HAudioDriverASIO classes to notify the user when an recording event occurs.

`#include <HaudioCallbacks.h>`

Public Member Functions

- virtual void OnDataReceived (int handle, HAudioData *data)

  Callback occuring when an audio driver receive a new audio buffer.

3.1.1 Detailed Description

This class is used by the HAudioDriverDirectX and HAudioDriverASIO classes to notify the user when an recording event occurs.

3.1.2 Member Function Documentation

3.1.2.1 virtual void CAudioCaptureCallback::OnDataReceived (int *handle*, HAudioData * *data*) [inline], [virtual]

Callback occuring when an audio driver receive a new audio buffer.

Parameters

handle	An handle to the audio divice
data	The audio buffer that was just received

3.2 CAudioPlaybackCallback Class Reference

This class is used by the HAudioDriverDirectX and HAudioDriverASIO classes to notify the user when a playback event occurs.

`#include <HaudioCallbacks.h>`

Public Member Functions

- virtual HAudioData * OnReadNewData (int handle)

  Callback occuring when an audio driver request a new audio buffer.
- virtual void OnAudioSourceEvent (int handle, eHAudioPlayEventCode code)

  Callback occuring when an audio player driver tigger some events.

3.2.1 Detailed Description

This class is used by the HAudioDriverDirectX and HAudioDriverASIO classes to notify the user when a playback event occurs.

3.2.2 Member Function Documentation

3.2.2.1 virtual HAudioData* CAudioPlaybackCallback::OnReadNewData (int *handle*) [inline],[virtual]

Callback occuring when an audio driver request a new audio buffer.

User override this function to fill audio data.

Returns

new audio data

Parameters

handle	An handle to the audio divic

3.2.2.2 virtual void CAudioPlaybackCallback::OnAudioSourceEvent (int *handle,* eHAudioPlayEventCode *code*) [inline],[virtual]

Callback occuring when an audio player driver tigger some events.

Parameters

handle	An handle to the audio divic
code	a trigger Event

3.3 CAudioPlayerCallback Class Reference

This class is used by the HAudioPlayer class to notify the user when an event occurs.

```
#include <HaudioCallbacks.h>
```

Public Member Functions

- virtual void OnPlayBuffer (HAudioData *data, unsigned __int64 index)

  Callback occuring when the player is starting to play an audio buffer.
- virtual void OnEndPlayback (bool endReached)

  Callback occuring when the end of the audio file is reached during playback.

3.3.1 Detailed Description

This class is used by the HAudioPlayer class to notify the user when an event occurs.

3.3.2 Member Function Documentation

3.3.2.1 virtual void CAudioPlayerCallback::OnPlayBuffer (HAudioData * *data*, unsigned __int64 *index*) [inline], [virtual]

Callback occuring when the player is starting to play an audio buffer.

Parameters

data	The audio buffer that is currently being played.
index	The index of the audio buffer.

3.3.2.2 virtual void CAudioPlayerCallback::OnEndPlayback (bool *endReached*) [inline], [virtual]

Callback occuring when the end of the audio file is reached during playback.

Parameters

endReached	The playback was stopped because the end was reached.

3.4 CAudioRecorderCallback Class Reference

This class is used by the HAudioRecorder class to notify the user when an event occurs.

```
#include <HaudioCallbacks.h>
```

Public Member Functions

- virtual void OnBufferReceived (HAudioData *data)

  Callback occuring when an audio buffer is successfully saved to an audio file.
- virtual void OnBufferSaved (unsigned __int64 index)

  Callback occuring when an audio buffer is successfully saved to an audio file.

3.4.1 Detailed Description

This class is used by the HAudioRecorder class to notify the user when an event occurs.

3.4.2 Member Function Documentation

3.4.2.1 virtual void CAudioRecorderCallback::OnBufferReceived (HAudioData * *data*) [inline], [virtual]

Callback occuring when an audio buffer is successfully saved to an audio file.

Parameters

data	The audio buffer received from the device.

3.4.2.2 virtual void CAudioRecorderCallback::OnBufferSaved (unsigned __int64 *index*) [inline], [virtual]

Callback occuring when an audio buffer is successfully saved to an audio file.

Parameters

index	The index in the audio file where the buffer was written. (1 being the first buffer)

3.5 CCineFormHDParam Struct Reference

Cineform HC CODEC parameter.

`#include <HAviCodecs.h>`

Inherits CCineFormParam.

Public Attributes

- bool ForceTo444

  Force 444 sampling.

3.5.1 Detailed Description

Cineform HC CODEC parameter.

3.6 CCineFormParam Struct Reference

The structure holding the the Cineform parameters.

`#include <HAviCodecs.h>`

Inherited by CCineFormHDParam, and CCineFormRawParam.

Public Attributes

- int EncoderQuality

  Encoder quality 0,Low Quality;1 Medium Quality;2 High Quality;3 FilmScan1;4 FilmScan2;5 FilmScan3;6 Uncompressed.
- float ColorMetrx [3][4]

  Color maxtrix.
- float Whitebalance [4]

  White balance.
- DWORD Buffer

  Encoding frame buffer.
- CineFormCurve CurveType

  Cinefrom curve.
- int Numerator

  Numerator for Gamma and log parameter.
- int Denominator

  Denominator for Gamma and log parameter.
- char LookFile [MAX_PATH]

  Iridas Look file.

3.6.1 Detailed Description

The structure holding the the Cineform parameters.

3.7 CCineFormRawParam Struct Reference

Cineform RAW CODEC parameter.

`#include <HAviCodecs.h>`

Inherits CCineFormParam.

Public Attributes

- DWORD BayerPattern

  Bayer pattern.

3.7.1 Detailed Description

Cineform RAW CODEC parameter.

3.8 CDaqSupportCallback Class Reference

This class manages the DAQ devices callbacks.

`#include <HDaqCallbacks.h>`

Public Member Functions

- virtual bool WantContinuousDataRead (int deviceIndex)

  This callback is used by the DAQ server that the client is able to receive DAQ data from a DAQ source.
- virtual bool OnDataReceived (int deviceIndex, HDaqDataSet *dataSet)

  This callback is used to provided to the client latest captured DAQ data set.
- virtual void OnDaqDeviceStatusChanged (int deviceIndex, int speed, double pollingTime, int range)

  This callback is used to notify the client a change in the device status occured.
- virtual bool WantContinuousDataRead (int deviceIndex)

  This callback is used by the DAQ server that the client is able to receive DAQ data from a DAQ source.
- virtual bool OnDataReceived (int deviceIndex, HDaqDataSet *dataSet)

  This callback is used to provided to the client latest captured DAQ data set.
- virtual void OnDaqDeviceStatusChanged (int deviceIndex, int speed, double pollingTime, int range)

  This callback is used to notify the client a change in the device status occured.

3.8.1 Detailed Description

This class manages the DAQ devices callbacks.

3.8.2 Member Function Documentation

3.8.2.1 virtual bool CDaqSupportCallback::WantContinuousDataRead (int *deviceIndex*) `[inline],[virtual]`

This callback is used by the DAQ server that the client is able to receive DAQ data from a DAQ source.

Returns

true will result in a call to OnDataReceived(). Otherwise ignored.

Parameters

deviceIndex	in> sender Device ID.

3.8.2.2 virtual bool CDaqSupportCallback::OnDataReceived (int *deviceIndex*, HDaqDataSet * *dataSet*) [inline], [virtual]

This callback is used to provided to the client latest captured DAQ data set.

Returns

true to continue receiving more DAQ data set.

Parameters

deviceIndex	in> sender Device ID.
dataSet	in> currerent DAQ data data set.

3.8.2.3 virtual void CDaqSupportCallback::OnDaqDeviceStatusChanged (int *deviceIndex*, int *speed*, double *pollingTime*, int *range*) [inline], [virtual]

This callback is used to notify the client a change in the device status occured.

New device settings are provided.

Parameters

deviceIndex	in> sender Device ID.
speed	in> Device sampling.
pollingTime	in> Device polling time.
range	in> DAQ data range ID.

3.8.2.4 virtual bool CDaqSupportCallback::WantContinuousDataRead (int *deviceIndex*) [inline], [virtual]

This callback is used by the DAQ server that the client is able to receive DAQ data from a DAQ source.

Returns

true will result in a call to OnDataReceived(). Otherwise ignored.

Parameters

deviceIndex	in> sender Device ID.

3.8.2.5 virtual bool CDaqSupportCallback::OnDataReceived (int *deviceIndex*, HDaqDataSet * *dataSet*) [inline], [virtual]

This callback is used to provided to the client latest captured DAQ data set.

Returns

true to continue receiving more DAQ data set.

Parameters

deviceIndex	in> sender Device ID.
dataSet	in> currerent DAQ data data set.

3.8.2.6 virtual void CDaqSupportCallback::OnDaqDeviceStatusChanged (int *deviceIndex*, int *speed*, double *pollingTime*, int *range*) [inline],[virtual]

This callback is used to notify the client a change in the device status occured.

New device settings are provided.

Parameters

deviceIndex	in> sender Device ID.
speed	in> Device sampling.
pollingTime	in> Device polling time.
range	in> DAQ data range ID.

3.9 CGPSDataStruct Struct Reference

GPS Data structure.

```
#include <GPSDataStruct.h>
```

Public Attributes

- float Longitude

  The longitude in degrees.
- float Latitude

  The latitude in degrees.
- float Altitude

  The altitude in meters.
- float reserved1

  Reserved.
- float reserved2

  Reserved.

3.9.1 Detailed Description

GPS Data structure.

3.10 HAACFile Class Reference

This class manages AAC audio files (Advance Audio Codec) AAC format can handle up to 5 audio channel.

```
#include <HAACFile.h>
```

Public Member Functions

- CHermesAAC * **GetAACFile** ()

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

FILE CREATE/ADDDATA/CLOSE FUNCTIONS

- bool Create (LPCTSTR aacPath)

  Creates an empty AAC file on disk, getting reading for storing audio samples.
- bool SetAudioParam (int sampleRate, int bitRate, int mapLen=0, int *maps=NULL)

  Set audio stream parameters.
- bool AddData (HAudioData *data)

  Append the specified audio data to the current ACC file.
- bool EndRecording ()

  Finish audio recording and switch to playback mode.
- bool Open (LPCTSTR aacPath)

  open a exist AAC file on disk, getting playback audio samples.
- bool GetAudioInfo (int &sampleRate, int &channels, int &bitDepth, int &bitrate, int &samplesPerBuffer)

  Get audio stream parameters.
- unsigned int GetTotalAudioDatas ()

  Retrieve the number of audio blocks stored in the current AAC.
- bool GetTimeStamp (unsigned __int64 index, long &timeStamp, unsigned short &timeStampMS, unsigned short &timeStampUS)

  Retrieve the timestamp of a given frame in the sequence.
- bool SetReadingPosition (unsigned __int64 position)

  Set the start position of the read.
- unsigned __int64 GetReadingPosition ()

  Retrieves the current read position, by returning sequence index of the current read position.
- unsigned __int64 GetPositionByTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS)

  Search the audio file to find the last audio that had been captured at a given time.
- bool ReadAudio (HAudioData *audioData)

  Read an audio data from the current AAC.
- bool Close ()

  Closes an open AAC File.

3.10.1 Detailed Description

This class manages AAC audio files (Advance Audio Codec) AAC format can handle up to 5 audio channel.

On a stereo (2 channels, left and right) set channelMap[2] where values should be [0]=0 [1]=1 for selecting channel 0 (left) and 1 (right). On a 4 channels input signal, storing only 2 channels, set channelMap[2] where values should be [0]=2 [1]=3 for selecting channel 2 and 3.

The typical (simplified) call order is :

```
//Saving captured images to a sequence file on disk
HAACFile aac;
aac.Create(_T("c:\\audio.aac"));
aac.SetAudioParam(48000,128000);
aac.AddData(audioDataA);
aac.AddData(audioDataB);
aac.AddData(audioDataC);
aac.AddData(audioDataD);
aac.AddData(audioDataE);
//and so on...
aac.Close();
```

3.10.2 Member Function Documentation

3.10.2.1 void HAACFile::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.10.2.2 eHErrorCode HAACFile::GetLastError ( )

See also

HGrabModule::GetLastError

3.10.2.3 bool HAACFile::Create (LPCTSTR *aacPath*)

Creates an empty AAC file on disk, getting reading for storing audio samples.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*aacPath*	The full path and file name where the audio is to be saved (ex: c:/audio.aac). Any AAC file with the same file name will be overwritten.

3.10.2.4 bool HAACFile::SetAudioParam (int *sampleRate*, int *bitRate*, int *mapLen* = 0, int * *maps* = NULL)

Set audio stream parameters.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*sampleRate*	Audio input sample rate. Should match the audio capture device settings.
in	*bitRate*	AAC compresion bit rate in bit per second.
in	*mapLen*	The number of items in "maps" table, corresponding to the number of audio channels to be stored.
in	*maps*	An array of input channels to be saved. The values of each cell in the array define a mapping for selecting the channels to be stored versus discarded.

3.10.2.5 bool HAACFile::AddData (HAudioData * *data*)

Append the specified audio data to the current ACC file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.10.2.6 bool HAACFile::EndRecording ( )

Finish audio recording and switch to playback mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.10.2.7 bool HAACFile::Open (LPCTSTR *aacPath*)

open a exist AAC file on disk, getting playback audio samples.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*aacPath*	The full path and file name where the audio is to open (ex: c:/audio.aac). Any aac file with the same name will be overwritten.

3.10.2.8 bool HAACFile::GetAudioInfo (int & *sampleRate,* int & *channels,* int & *bitDepth,* int & *bitrate,* int & *samplesPerBuffer*)

Get audio stream parameters.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*sampleRate*	audio sample rate. Should match audio device settings.
out	*channels*	number of audio channels.
out	*bitDepth*	audio bitDepth
out	*bitrate*	compression bit rate in bit per second.
out	*samplesPer↩ Buffer*	sample per each audio buffer

3.10.2.9 unsigned int HAACFile::GetTotalAudioDatas ( )

Retrieve the number of audio blocks stored in the current AAC.

Returns

The number of audio blocks in the AAC.

3.10.2.10 bool HAACFile::GetTimeStamp (unsigned __int64 *index,* long & *timeStamp,* unsigned short & *timeStampMS,* unsigned short & *timeStampUS*)

Retrieve the timestamp of a given frame in the sequence.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The audio data index.
out	*timeStamp*	The absolute timestamp in seconds.
out	*timeStampMS*	The millisecond part of the timestamp.
out	*timeStampUS*	The microsecond part of the timestamp.

3.10.2.11 bool HAACFile::SetReadingPosition (unsigned __int64 *position*)

Set the start position of the read.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

position	index [in] The desired playback position.

3.10.2.12 unsigned __int64 HAACFile::GetReadingPosition ( )

Retrieves the current read position, by returning sequence index of the current read position.

Returns

The current read position.

3.10.2.13 unsigned __int64 HAACFile::GetPositionByTimeStamp (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Search the audio file to find the last audio that had been captured at a given time.

Returns

the audio data index.

Parameters

in	*timeStamp*	The absolute timestamp in seconds.
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.10.2.14 bool HAACFile::ReadAudio (HAudioData * *audioData*)

Read an audio data from the current AAC.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.10.2.15 bool HAACFile::Close ( )

Closes an open AAC File.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.11 HAudioData Class Reference

An audio buffer holding a range of samples.

```
#include <HAudioData.h>
```

Public Member Functions

- bool ReallocateAudio (eHAudioFormat format, int channels, int samples, long audioDepth, long audioDepth↩ Real, long sampleRate)

  Reallocate memory in an HAudioData.
- BYTE * GetAudioDataBuffer ()

  Get a pointer to the audio buffer.
- int GetSampleCount ()

  Get the number of audio samples in the HAudioData.
- int GetChannelNum ()

  Get the number of channels in the HAudioData.
- int GetSamplerate ()

  Get the number of sample rate in the HAudioData.
- eHAudioFormat GetFormat ()

  Get the format of the HAudioData.
- int GetAudioBitDepth ()

  Get the bit depth of the HAudioData.
- int GetAudioBitDepthReal ()

  Get the real bit depth of the HAudioData.
- long GetTimeStamp ()

  Get the time stamp of the HAudiodata.
- unsigned short GetTimeStampMS ()

  This will retrieve the milliseconds part of the absolute time stamp.
- unsigned short GetTimeStampUS ()

  The microseconds part of the absolute time stamp.
- void SetTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS=0)

  Change the time stamp of an HAudioData.
- eHAudioCompression GetCompressionFormat ()

  Get compression type.
- int GetCompressedChannelSizeBytes ()

  Get compression buffer size.
- void * GetCompressedChannel ()

  Get compression buffer pointer.
- int GetCompressedImageSizeBytes ()

  Get real compression data size.
- void * GetCompressedImage ()

  Get real compression data pointer.
- void FreeCompressedChannel ()

  Free compression buffer.

Static Public Member Functions

STATIC FUNCTIONS

- static HAudioData * CreateAudioData ()
  Create an empty HAudioData.
- static HAudioData * CloneAudioData (HAudioData *srcAudioData)
  Create a perfect copy of a source HAudioData.
- static void CloneAudioData (HAudioData *srcAudioData, HAudioData *destAudioData)
  Creates a perfect copy of the srcAudioData in the destAudioData.
- static void DeleteAudioData (HAudioData *audioData)
  Delete an HAudioData that was created with a call to CreateHAudioData.

3.11.1 Detailed Description

An audio buffer holding a range of samples.

3.11.2 Member Function Documentation

3.11.2.1 static HAudioData* HAudioData::CreateAudioData ( ) [static]

Create an empty HAudioData.

When it is no longer needed, free it by using DeleteHAudioData.

Returns

A pointer to a newly created HAudioData.

3.11.2.2 static HAudioData* HAudioData::CloneAudioData (HAudioData * *srcAudioData*) [static]

Create a perfect copy of a source HAudioData.

When it is no longer needed, free it by using DeleteHAudioData.

Returns

A pointer to a newly created HAudioData.

Parameters

in	*srcAudioData*	The source AudioData to clone.

3.11.2.3 static void HAudioData::CloneAudioData (HAudioData * *srcAudioData*, HAudioData * *destAudioData*) [static]

Creates a perfect copy of the srcAudioData in the destAudioData.

For the destAudioData set, only use AudioData you created using HAudioData::CreateHAudioData()

Parameters

in	*srcAudioData*	The AudioData source to clone.
in	*destAudioData*	The destination AudioData.

3.11.2.4 bool HAudioData::ReallocateAudio (eHAudioFormat *format*, int *channels*, int *samples*, long *audioDepth*, long *audioDepthReal*, long *sampleRate*)

Reallocate memory in an HAudioData.

Delete the HAudioData currently stored in the HAudioData and will reallocate memory to fit the new HAudioData according the specified parameters. WARNING : Don't reallocate an buffer that you did not create with CreateH↩ AudioData()

Parameters

in	*format*	The audio format.
in	*channels*	The number of channels.
in	*samples*	The number of samples per channel.
in	*audioDepth*	The bit depth of the AudioData.
in	*audioDepthReal*	The real bit depth of the AudioData.
in	*sampleRate*	The sample rate of the AudioData.

3.11.2.5 BYTE* HAudioData::GetAudioDataBuffer ( )

Get a pointer to the audio buffer.

Returns

The pointer to an audio buffer (the actual audio bytes).

3.11.2.6 int HAudioData::GetSampleCount ( )

Get the number of audio samples in the HAudioData.

Returns

The number of samples.

3.11.2.7 int HAudioData::GetChannelNum ( )

Get the number of channels in the HAudioData.

Returns

The number of channels.

3.11.2.8 int HAudioData::GetSamplerate ( )

Get the number of sample rate in the HAudioData.

Returns

The number of sample rate.

3.11.2.9 eHAudioFormat HAudioData::GetFormat ( )

Get the format of the HAudioData.

Returns

The function will return the format of the AudioData.

3.11.2.10 int HAudioData::GetAudioBitDepth ( )

Get the bit depth of the HAudioData.

Returns

The function will return the bit depth of the HAudioData. If the HAudioData is empty, the returned value is 0.

3.11.2.11 int HAudioData::GetAudioBitDepthReal ( )

Get the real bit depth of the HAudioData.

Returns

The function will return the real bit depth of the HAudioData. If the AudioData is empty, the returned value is 0.

3.11.2.12 long HAudioData::GetTimeStamp ( )

Get the time stamp of the HAudiodata.

Returns

The absolute time stamp in seconds.

3.11.2.13 unsigned short HAudioData::GetTimeStampMS ( )

This will retrieve the milliseconds part of the absolute time stamp.

Depending on the latency of the operating system, the precision might be off by 1 to 3 milliseconds.

Returns

The milliseconds part of the time stamp.

3.11.2.14 unsigned short HAudioData::GetTimeStampUS ( )

The microseconds part of the absolute time stamp.

This value will be 0 unless the frame was time stamped while using an external high performance timer device.

Returns

The microsecond part of the time stamp.

3.11.2.15 void HAudioData::SetTimeStamp (long *timeStamp*, unsigned short *timeStampMS*, unsigned short *timeStampUS* = 0)

Change the time stamp of an HAudioData.

Parameters

in	*timeStamp*	The absolute time stamp in seconds.
in	*timeStampMS*	The milliseconds part of the time stamp.
in	*timeStampUS*	The microsecond part of the time stamp.

3.12 HAudioDriverASIO Class Reference

Manage an ASIO sound device.

`#include <HAudioDriverASIO.h>`

Public Member Functions

- HAudioDriverASIO (LPCTSTR NpxASIODLLPath)

  Constructor.
- bool GetDriverName (LPTSTR name)

  Retrieves the audio source device and driver name.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

INPUT FUNCTIONS

- bool AudioInputPresent ()

  Checks if the audio device is available and ready for connection.
- int GetAudioInputDeviceCount ()

  Get the number of audio input devices.
- bool GetAudioInputDeviceName (int deviceIndex, LPTSTR name)

  Get the name of the audio device.
- int OpenAudioInputDevice (int &deviceIndex)

  Open an input audio device.
- bool CloseAudioInputDevice (int handle)

  Close an open audio input device.
- bool SetAudioInputTimeSourceDLL (int handle, LPCTSTR dllPath, int timeSourceOffsetUS)

  Load an external time source for timestamping the AudioData.
- int GetAudioInputSampleRateInfoCount (int handle)

  Get the numbers of sample rate settings supported by the input device.
- bool GetAudioInputSampleRateInfo (int handle, int index, int ∗sampleRateID, LPTSTR name, int ∗value)

  Get info on each sample rate setting.
- bool OpenAudioInputHardwareSettingsDlg (int handle)

  Try to open the audio input device hardware settings (if any).
- int GetAudioInputSampleRate (int handle)

  Get the audio input sample rate (in samples per second)
- int GetAudioInputChannelCount (int handle)

  Get the numbers of input channels on the device.
- bool GetAudioInputChannelInfoByChannelIndex (int handle, int index, int ∗channelID, LPTSTR name, eHAudioFormat ∗format, int ∗depth, int ∗depthReal)

  Get information on each channel's format.
- bool StartAudioInputCapture (int handle, int &sampleRate, CAudioCaptureCallback ∗captureCallback, int mapLen, int ∗channelMap)

  Start capturing audio from the audio input device.
- bool StopAudioInputCapture (int handle)

  Stop capturing audio.

- bool IsAudioInputCapturing (int handle)
  Check if audio input device is capturing.
- void SetCaptureBufferCount (int buffer)
  Set the number of buffers to be used by the Audio capure mechanism.
- int GetCaptureBufferCount ()
  Get the number of buffers used by the Audio capure mechanism.
- int GetCaptureBufferUsage ()
  Get the number of frames waiting to be processed by the OnDataReceived.
- int GetCaptureMaxBufferUsage ()
  Get the maximum number of buffers reached since acquisition started.
- void ResetCaptureBufferUsage ()
  Get the total number of frames allocated for capture.

OUTPUT FUNCTIONS

- bool AudioOutputPresent ()
  Check if an ouptut audio device is available.
- int GetAudioOutputDeviceCount ()
  Get number of physical devices available for audio output.
- bool GetAudioOutputDeviceName (int deviceIndex, LPTSTR name)
  Get the name of the audio output device.
- int OpenAudioOutputDevice (int &deviceIndex)
  Open an audio output device.
- bool CloseAudioOutputDevice (int handle)
  Close an output audio device.
- int GetAudioOutputSamplePerBuffer (int handle)
  Get the number of samples per buffer requrired by the audio output device.
- int GetAudioOutputSampleRateInfoCount (int handle)
  Get the numbers of sample rate settings supported by the output device.
- bool GetAudioOutputSampleRateInfo (int handle, int index, int ∗sampleRateID, LPTSTR name, int ∗value)
  Get info on each sample rate setting.
- bool OpenAudioOutputHardwareSettingsDlg (int handle)
  Open the audio output hardware settings (if available).
- int GetAudioOutputSampleRate (int handle)
  Get current sample rate value.
- int GetAudioOutputChannelCount (int handle)
  Get numbers of output channels on the device.
- bool GetAudioOutputChannelInfoByChannelIndex (int handle, int index, int ∗channelID, LPTSTR name, eHAudioFormat ∗format, int ∗depth, int ∗depthReal)
  Get an output channel settings by index.
- bool StartAudioOutputPlayback (int handle, int &sampleRate, CAudioPlaybackCallback ∗readCallback, int mapLen, int ∗channelMap)
  Start audio playback.
- bool StopAudioOutputPlayback (int handle)
  Stop audio playback.
- bool IsAudioOutputPlaying (int handle)
  Check if the device is playing.

3.12.1 Detailed Description

Manage an ASIO sound device.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 HAudioDriverASIO::HAudioDriverASIO (LPCTSTR *NpxASIODLLPath*)

Constructor.

Parameters

in	*NpxASIODLL↩ Path*	The full path to NpxAsio.dll

3.12.3 Member Function Documentation

3.12.3.1 void HAudioDriverASIO::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.12.3.2 eHErrorCode HAudioDriverASIO::GetLastError ( )

See also

HGrabModule::GetLastError

3.12.3.3 bool HAudioDriverASIO::GetDriverName (LPTSTR *name*)

Retrieves the audio source device and driver name.

Returns

True if the function is successful or false if the function failed. In case of failure, the driver DLL missing.

Parameters

out	*name*	The name of the driver.

3.12.3.4 bool HAudioDriverASIO::AudioInputPresent ( )

Checks if the audio device is available and ready for connection.

Make sure the device is present before calling any other functions in this class.

Returns

True if input can be used.

3.12.3.5 int HAudioDriverASIO::GetAudioInputDeviceCount ( )

Get the number of audio input devices.

Returns

Input device count.

3.12.3.6 bool HAudioDriverASIO::GetAudioInputDeviceName (int *deviceIndex,* LPTSTR *name*)

Get the name of the audio device.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index [0, GetAudioInputDeviceCount()-1]
in,out	*name*	The name of the audio input device. Allocate TCHAR [MAX_PATH].

3.12.3.7 int HAudioDriverASIO::OpenAudioInputDevice (int & *deviceIndex*)

Open an input audio device.

Returns the "handle" required by other the other input functions.

Returns

The handle or -1 if the function failed.

Parameters

in,out	*deviceIndex*	the device index to open. If an ASIO driver is already open, return that device index.

3.12.3.8 bool HAudioDriverASIO::CloseAudioInputDevice (int *handle*)

Close an open audio input device.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.12.3.9 bool HAudioDriverASIO::SetAudioInputTimeSourceDLL (int *handle,* LPCTSTR *dllPath,* int *timeSourceOffsetUS*)

Load an external time source for timestamping the AudioData.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*handle*	the device handle.
in	*dllPath*	The path to the time source dll.
in	*timeSource↩ OffsetUS*	A time offset in microseconds to apply to the time source (if any)

3.12.3.10 int HAudioDriverASIO::GetAudioInputSampleRateInfoCount (int *handle*)

Get the numbers of sample rate settings supported by the input device.

Returns

The number of sample rate settings.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.12.3.11 bool HAudioDriverASIO::GetAudioInputSampleRateInfo (int *handle*, int *index*, int * *sampleRateID*, LPTSTR *name*, int * *value*)

Get info on each sample rate setting.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice().
in	*index*	The sample rate settings index. [0, GetAudioInputSampleRateInfoCount()-1]
out	*sampleRateID*	The identifier for this sample rate setting.
out	*name*	A string representation of the sample rate.
out	*value*	The sample rate (in samples per second)

3.12.3.12 bool HAudioDriverASIO::OpenAudioInputHardwareSettingsDlg (int *handle*)

Try to open the audio input device hardware settings (if any).

Depending on the manufacturer driver implementation, this function call might not show anything.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.12.3.13 int HAudioDriverASIO::GetAudioInputSampleRate (int *handle*)

Get the audio input sample rate (in samples per second)

Returns

The sample rate.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.12.3.14 int HAudioDriverASIO::GetAudioInputChannelCount (int *handle*)

Get the numbers of input channels on the device.

Returns

channel count.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.12.3.15 bool HAudioDriverASIO::GetAudioInputChannelInfoByChannelIndex (int *handle*, int *index*, int * *channelID*, LPTSTR *name*, **eHAudioFormat** * *format*, int * *depth*, int * *depthReal*)

Get information on each channel's format.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()
in	*index*	The channel index [0, GetAudioInputChannelCount()-1]
out	*channelID*	The identifier of the channel.
out	*name*	A string representation of the channel name.
out	*format*	The audio format of the channel.
out	*depth*	The bit depth of the channel.
out	*depthReal*	The real bit depth of the channel.

3.12.3.16 bool HAudioDriverASIO::StartAudioInputCapture (int *handle*, int & *sampleRate*, **CAudioCaptureCallback** * *captureCallback*, int *mapLen*, int * *channelMap*)

Start capturing audio from the audio input device.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()
in,out	*sampleRate*	The sample rate.
in	*captureCallback*	Callback for Audio data capture
in	*mapLen*	The number of items in ìmapsî.
in	*channelMap*	An array of channels to capture from. For example, if you want to convert from channels 1,3 & 4, the maps array would be [1,3,4]. Use NULL to use the default channel map.

3.12.3.17 bool HAudioDriverASIO::StopAudioInputCapture (int *handle*)

Stop capturing audio.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.12.3.18 bool HAudioDriverASIO::IsAudioInputCapturing (int *handle*)

Check if audio input device is capturing.

Returns

True if t audio input device is capturing, false otherwise.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.12.3.19 void HAudioDriverASIO::SetCaptureBufferCount (int *buffer*)

Set the number of buffers to be used by the Audio capure mechanism.

Having a large number of buffers will help to prevent frame drops.

3.12.3.20 int HAudioDriverASIO::GetCaptureBufferUsage ( )

Get the number of frames waiting to be processed by the OnDataReceived.

callback.

3.12.3.21 void HAudioDriverASIO::ResetCaptureBufferUsage ( )

Get the total number of frames allocated for capture.

Returns

The total number of frames allocated for acquisition.

3.12.3.22 bool HAudioDriverASIO::AudioOutputPresent ( )

Check if an ouptut audio device is available.

Make sure the device is present before calling any other functions in this class.

Returns

True if the audio output device can be used.

3.12.3.23 int HAudioDriverASIO::GetAudioOutputDeviceCount ( )

Get number of physical devices available for audio output.

Returns

The output device count.

3.12.3.24 bool HAudioDriverASIO::GetAudioOutputDeviceName (int *deviceIndex*, LPTSTR *name*)

Get the name of the audio output device.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*deviceIndex*	The device index [0, GetAudioOutputDeviceCount()-1]
in,out	*name*	The name of the audio input device. Allocate TCHAR [MAX_PATH].

3.12.3.25 int HAudioDriverASIO::OpenAudioOutputDevice (int & *deviceIndex*)

Open an audio output device.

Returns the "handle" required by other the other output functions.

Returns

The handle or -1 if the function failed.

Parameters

in	*deviceIndex*	The device index to open.

3.12.3.26 bool HAudioDriverASIO::CloseAudioOutputDevice (int *handle*)

Close an output audio device.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle.

3.12.3.27 int HAudioDriverASIO::GetAudioOutputSamplePerBuffer (int *handle*)

Get the number of samples per buffer requrired by the audio output device.

Returns

The number of samples per buffer.

Parameters

in	*handle*	The device handle.

3.12.3.28 int HAudioDriverASIO::GetAudioOutputSampleRateInfoCount (int *handle*)

Get the numbers of sample rate settings supported by the output device.

Returns

The number of sample rate settings.

Parameters

in	*handle*	the device handle from OpenAudioOutputDevice()

3.12.3.29 bool HAudioDriverASIO::GetAudioOutputSampleRateInfo (int *handle*, int *index*, int * *sampleRateID*, LPTSTR *name*, int * *value*)

Get info on each sample rate setting.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice().
in	*index*	The sample rate settings index. [0, GetAudioOutputSampleRateInfoCount()-1].
out	*sampleRateID*	The identifier for this sample rate setting.
out	*name*	A string representation of the sample rate.
out	*value*	The sample rate (in samples per second)

3.12.3.30 bool HAudioDriverASIO::OpenAudioOutputHardwareSettingsDlg (int *handle*)

Open the audio output hardware settings (if available).

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()

3.12.3.31 int HAudioDriverASIO::GetAudioOutputSampleRate (int *handle*)

Get current sample rate value.

Returns

The current sample rate value.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()

3.12.3.32 int HAudioDriverASIO::GetAudioOutputChannelCount (int *handle*)

Get numbers of output channels on the device.

Returns

The channel count.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()

3.12.3.33 bool HAudioDriverASIO::GetAudioOutputChannelInfoByChannelIndex (int *handle*, int *index*, int * *channelID*, LPTSTR *name*, **eHAudioFormat** * *format*, int * *depth*, int * *depthReal*)

Get an output channel settings by index.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice().
in	*index*	The channel index [0, GetAudioOutputChannelCount()-1].
out	*channelID*	The channel identifer.
out	*name*	The channel name.
out	*format*	The audio format of the channel.
out	*depth*	The bit depth of the channel.
out	*depthReal*	The real bit depth of the channel.

3.12.3.34 bool HAudioDriverASIO::StartAudioOutputPlayback (int *handle*, int & *sampleRate*, **CAudioPlaybackCallback** * *readCallback*, int *mapLen*, int * *channelMap*)

Start audio playback.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()
in,out	*sampleRate*	The sample rate to use.
in	*readCallback*	Callback for audio playback
in	*mapLen*	The number of items in ìmapsî.
in	*channelMap*	An array of channels to play from. For example, if you want to convert from channels 1,3 & 4, the maps array would be [1,3,4]. Use NULL to use the default channel map.

3.12.3.35 bool HAudioDriverASIO::StopAudioOutputPlayback (int *handle*)

Stop audio playback.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()

3.12.3.36 bool HAudioDriverASIO::IsAudioOutputPlaying (int *handle*)

Check if the device is playing.

Returns

True if the audio output device is playing, false otherwise.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice().

3.13 HAudioDriverDirectX Class Reference

Manage a DirectX sound device.

```
#include <HAudioDriverDirectX.h>
```

Public Member Functions

- HAudioDriverDirectX (LPCTSTR NpxDirectXAudioDLLPath)

  Constructor.
- bool GetDriverName (LPTSTR name)

  Retrieves the audio source device and driver name.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

INPUT FUNCTIONS

- bool AudioInputPresent ()

  Check if the audio device is available and connected.
- int GetAudioInputDeviceCount ()

  Get the number of audio input devices.
- bool GetAudioInputDeviceName (int deviceIndex, LPTSTR name)

  Get the name of the audio device.
- int OpenAudioInputDevice (int deviceIndex)

  Open an input audio device.
- bool CloseAudioInputDevice (int handle)

  Close an input audio device.
- bool SetAudioInputTimeSourceDLL (int handle, LPCTSTR dllPath, int timeSourceOffsetUS)

  Load an external time source for timestamping the AudioData.
- int GetAudioInputFormatCount (int handle)

  Get the number of input formats supported.
- bool GetAudioInputFormatInfo (int handle, int index, LPTSTR name, eHAudioFormat *format, int *bits↩PerSample, int *channels, int *sampleRate)

  Retrieve the device capabilities for a specific input format.
- bool SetAudioInputFormat (int handle, int index, DWORD channelMask, int samplesPerBuffer)

  Set the input format settings used for data capture.
- bool GetAudioInputFormat (int handle, int &index, DWORD &channelMask, int &samplesPerBuffer)

Get current input format settings used for data capture.

- bool StartAudioInputCapture (int handle, int &sampleRate, CAudioCaptureCallback *captureCallback)

  Start the audio capture.
- bool StopAudioInputCapture (int handle)

  Stop the audio capture.
- bool IsAudioInputCapturing (int handle)

  Check if audio input device is currently capturing data.

BUFFERS FUNCTIONS

- void SetCaptureBufferCount (int buffer)

  Set the number of buffers to be used by the Audio capure mechanism.
- int GetCaptureBufferCount ()

  Get the number of buffers used by the Audio capure mechanism.
- int GetCaptureBufferUsage ()

  Get the number of frames waiting to be processed by the OnDataReceived.
- int GetCaptureMaxBufferUsage ()

  Get the maximum number of buffers reached since acquisition started.
- void ResetCaptureBufferUsage ()

  Get the total number of frames allocated for capture.

OUTPUT FUNCTIONS

- bool AudioOutputPresent ()

  Check if an output audio device is available.
- int GetAudioOutputDeviceCount ()

  Get number of physical devices available for audio output.
- bool GetAudioOutputDeviceName (int deviceIndex, LPTSTR name)

  Get the name of the audio output device.
- int OpenAudioOutputDevice (int &deviceIndex)

  Open an audio output device.
- bool CloseAudioOutputDevice (int handle)

  Close an output audio device.
- int GetAudioOutputSampleBufferSize (int handle)

  Get the number of samples per buffer requrired by the audio output device.
- int GetAudioOutputFormatCount (int handle)

  Get the number of output formats supported by output device.
- bool GetAudioOutputFormatInfo (int handle, int index, LPTSTR name, eHAudioFormat *format, int *bits↩PerSample, int *channels, int *sampleRate)

  Retrieve the device capabilities for a specific output format.
- bool SetAudioOutputFormat (int handle, HWND window, int index, DWORD channelMask, int samples↩PerBuffer)

  Set the output format used for recording.
- bool GetAudioOutputFormat (int handle, int &index, DWORD &channelMask, int &samplesPerBuffer)

  Get current output format settings for audio playback.
- bool StartAudioOutputPlayback (int handle, CAudioPlaybackCallback *readCallback, int mapLen=0, int *channelMap=NULL)

  Start playing an audio stream to an audio output device.
- bool StopAudioOutputPlayback (int handle)

  Stop playing an audio stream.
- bool IsAudioOutputPlaying (int handle)

  Check if audio output device is currently playing.
- bool SetOutputVolume (int handle, int percentage)

  Set the audio output volume level.
- int GetOutputVolume (int handle)

  Get the audio output volume level.

3.13.1 Detailed Description

Manage a DirectX sound device.

3.13.2 Constructor & Destructor Documentation

3.13.2.1 HAudioDriverDirectX::HAudioDriverDirectX (LPCTSTR *NpxDirectXAudioDLLPath*)

Constructor.

Parameters

in	*NpxDirectX↩ AudioDLLPath*	The full path to NpxDirectXAudio.dll

3.13.3 Member Function Documentation

3.13.3.1 void HAudioDriverDirectX::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.13.3.2 eHErrorCode HAudioDriverDirectX::GetLastError ( )

See also

HGrabModule::GetLastError

3.13.3.3 bool HAudioDriverDirectX::GetDriverName (LPTSTR *name*)

Retrieves the audio source device and driver name.

Returns

True if the function is successful or false if the function failed. In case of failure, the driver DLL missing.

Parameters

out	*name*	the name of the driver.

3.13.3.4 bool HAudioDriverDirectX::AudioInputPresent ( )

Check if the audio device is available and connected.

Make sure the device is present before calling any other input functions.

Returns

True if the audio input can be used.

3.13.3.5 int HAudioDriverDirectX::GetAudioInputDeviceCount ( )

Get the number of audio input devices.

Returns

The Audio input device count.

3.13.3.6 bool HAudioDriverDirectX::GetAudioInputDeviceName (int *deviceIndex,* LPTSTR *name*)

Get the name of the audio device.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index [0, GetAudioInputDeviceCount()-1]
in,out	*name*	The name of the audio input device. Allocate TCHAR [MAX_PATH].

3.13.3.7 int HAudioDriverDirectX::OpenAudioInputDevice (int *deviceIndex*)

Open an input audio device.

Returns the "handle" required by other the other input functions.

Returns

The handle or -1 if the function failed.

Parameters

in	*deviceIndex*	the device index to open

3.13.3.8 bool HAudioDriverDirectX::CloseAudioInputDevice (int *handle*)

Close an input audio device.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.13.3.9 bool HAudioDriverDirectX::SetAudioInputTimeSourceDLL (int *handle,* LPCTSTR *dllPath,* int *timeSourceOffsetUS*)

Load an external time source for timestamping the AudioData.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()
in	*dllPath*	The path to the time source dll.
in	*timeSource↩ OffsetUS*	A time offset in microseconds to apply to the time source (if any)

3.13.3.10 int HAudioDriverDirectX::GetAudioInputFormatCount (int *handle*)

Get the number of input formats supported.

For example, Stereo/16-bit/44100 is one format.

Returns

The input format count.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.13.3.11 bool HAudioDriverDirectX::GetAudioInputFormatInfo (int *handle,* int *index,* LPTSTR *name,* **eHAudioFormat** ∗ *format,* int ∗ *bitsPerSample,* int ∗ *channels,* int ∗ *sampleRate*)

Retrieve the device capabilities for a specific input format.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice().
in	*index*	The input format index [0, GetAudioInputFormatCount()-1].
out	*name*	The format name.
out	*format*	The audio format encoding.
out	*bitsPerSample*	The bits per samples (8 or 16).
out	*channels*	The channels (H_CHANNEL_LEFT, H_CHANNEL_RIGHT or H_CHANNEL↩ _STEREO).
out	*sampleRate*	The sample rate.

3.13.3.12 bool HAudioDriverDirectX::SetAudioInputFormat (int *handle,* int *index,* DWORD *channelMask,* int *samplesPerBuffer*)

Set the input format settings used for data capture.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice().
in	*index*	The audio format index.
in	*channelMask*	The channels mask to capture from.
in	*samplesPer↩ Buffer*	The number of samples in each AudioData.

3.13.3.13 bool HAudioDriverDirectX::GetAudioInputFormat (int *handle*, int & *index*, DWORD & *channelMask*, int & *samplesPerBuffer*)

Get current input format settings used for data capture.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice().
out	*index*	The audio format index.
out	*channelMask*	The channels mask to capture from.
out	*samplesPer↩ Buffer*	The number of samples in each AudioData.

3.13.3.14 bool HAudioDriverDirectX::StartAudioInputCapture (int *handle*, int & *sampleRate*, CAudioCaptureCallback * *captureCallback*)

Start the audio capture.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()
in,out	*sampleRate*	The desired sample rate for the capture.
in	*captureCallback*	Callback for Audio data capture

3.13.3.15 bool HAudioDriverDirectX::StopAudioInputCapture (int *handle*)

Stop the audio capture.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.13.3.16 bool HAudioDriverDirectX::IsAudioInputCapturing (int *handle*)

Check if audio input device is currently capturing data.

Returns

True if capturing, false otherwise.

Parameters

in	*handle*	The device handle from OpenAudioInputDevice()

3.13.3.17 void HAudioDriverDirectX::SetCaptureBufferCount (int *buffer*)

Set the number of buffers to be used by the Audio capure mechanism.

Having a large number of buffers will help to prevent frame drops.

3.13.3.18 int HAudioDriverDirectX::GetCaptureBufferUsage ( )

Get the number of frames waiting to be processed by the OnDataReceived.

callback.

3.13.3.19 void HAudioDriverDirectX::ResetCaptureBufferUsage ( )

Get the total number of frames allocated for capture.

Returns

The total number of frames allocated for acquisition.

3.13.3.20 bool HAudioDriverDirectX::AudioOutputPresent ( )

Check if an output audio device is available.

Make sure the device is present before calling any other audio output functions.

Returns

True if audio output can be used.

3.13.3.21 int HAudioDriverDirectX::GetAudioOutputDeviceCount ( )

Get number of physical devices available for audio output.

Returns

The output device count.

3.13.3.22 bool HAudioDriverDirectX::GetAudioOutputDeviceName (int *deviceIndex,* LPTSTR *name*)

Get the name of the audio output device.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*deviceIndex*	The device index [0, GetAudioOutputDeviceCount()-1]
in,out	*name*	The name of the audio input device.

3.13.3.23 int HAudioDriverDirectX::OpenAudioOutputDevice (int & *deviceIndex*)

Open an audio output device.

Returns the "handle" required by other the other output functions.

Returns

The handle or -1 if the function failed.

Parameters

in	*deviceIndex*	the device index to open.

3.13.3.24 bool HAudioDriverDirectX::CloseAudioOutputDevice (int *handle*)

Close an output audio device.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle.

3.13.3.25 int HAudioDriverDirectX::GetAudioOutputSampleBufferSize (int *handle*)

Get the number of samples per buffer requrired by the audio output device.

Returns

The number of samples per buffer.

Parameters

in	*handle*	The device handle.

3.13.3.26 int HAudioDriverDirectX::GetAudioOutputFormatCount (int *handle*)

Get the number of output formats supported by output device.

Returns

The output format count.

Parameters

in	*handle*	the device handle.

3.13.3.27 bool HAudioDriverDirectX::GetAudioOutputFormatInfo (int *handle*, int *index*, LPTSTR *name*, eHAudioFormat * *format*, int * *bitsPerSample*, int * *channels*, int * *sampleRate*)

Retrieve the device capabilities for a specific output format.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()
in	*index*	The output format index [0, GetAudioOutputFormatCount()-1]
out	*name*	The name of the output format
out	*format*	The format name
out	*bitsPerSample*	The bits pen samples (8 or 16).
out	*channels*	The channels (H_CHANNEL_LEFT, H_CHANNEL_RIGHT or H_CHANNEL↩ _STEREO).
out	*sampleRate*	The sample rate.

3.13.3.28 bool HAudioDriverDirectX::SetAudioOutputFormat (int *handle*, HWND *window*, int *index*, DWORD *channelMask*, int *samplesPerBuffer*)

Set the output format used for recording.

Window handle use to bunding with the device and interact with other instances of the device. Set the output format used for audio playback.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	the device handle from OpenAudioOutputDevice()
in	*window*	A HWND (window's handle);
in	*index*	The audio format index
in	*channelMask*	Mask of channels used for playback.
in	*samplesPer↩ Buffer*	The number of samples in each AudioData

3.13.3.29 bool HAudioDriverDirectX::GetAudioOutputFormat (int *handle*, int & *index*, DWORD & *channelMask*, int & *samplesPerBuffer*)

Get current output format settings for audio playback.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()
out	*index*	The audio format index.
out	*channelMask*	Mask of channels used for playback.
in	*samplesPer↩ Buffer*	The number of samples required in each AudioData

3.13.3.30 bool HAudioDriverDirectX::StartAudioOutputPlayback (int *handle*, CAudioPlaybackCallback * *readCallback*, int *mapLen* = 0, int * *channelMap* = NULL)

Start playing an audio stream to an audio output device.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()
in	*readCallback*	Callback for audio playback
in	*mapLen*	The number of items in maps (0,1,2).
in	*channelMap*	An array of channels to play from. For example, if you want to convert from channels 1,3 & 4, the maps array would be [1,3,4]. Use NULL to use the default channel map.

3.13.3.31 bool HAudioDriverDirectX::StopAudioOutputPlayback (int *handle*)

Stop playing an audio stream.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()

3.13.3.32 bool HAudioDriverDirectX::IsAudioOutputPlaying (int *handle*)

Check if audio output device is currently playing.

Returns

True if the audio output device is playing, false otherwise.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()

3.13.3.33 bool HAudioDriverDirectX::SetOutputVolume (int *handle*, int *percentage*)

Set the audio output volume level.

Returns

True if the function is successful of false if the function failed.

Parameters

in	*handle*	The device handle from OpenAudioOutputDevice()
in	*percentage*	Volume percentage to set [0,100]

3.13.3.34 int HAudioDriverDirectX::GetOutputVolume (int *handle*)

Get the audio output volume level.

[0,100]

Returns

Volume percentage

Parameters

in	*handle*	the device handle from OpenAudioOutputDevice()

3.14 HAudioFile Class Reference

This class is used to manage the creation/reading/writing of an audio file.

```
#include <HAudioFile.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

AUDIO INFO GETTERS

- long GetAudioSamplesPerBuffer ()
  Retrieves Samples size stored in the current audio file.
- long GetAudioChannelCount ()
  Retrieves numbers of channels stored in the current audio file.
- eHAudioFormat GetAudioFormat ()
  Retrieves audio format stored in the current audio file.
- long GetAudioDepth ()
  Retrieves the bit depth of the audio stored in the current audio file.
- long GetAudioDepthReal ()
  Retrieves the real bit depth of the audio stored in the current audio file.
- int GetSampleRate ()
  Retrieves the sample rate stored in the current audio file.
- void SetSampleRate (int rate)
  Modifies sample rate stored in the current audio file.
- bool GetAufioFileType (eAudioFileFormat &format)
  Retrieves audio file format in the current audio file.

FILE CREATE/OPEN/CLOSE FUNCTIONS

- bool Create (eAudioFileFormat fileType, LPCTSTR audioPath, eAudFileMode audoMode=H_AUD_FIL↩E_AUD_BLOCK)
  Creates an empty sequence file on disk or in RAM.

- bool Open (LPCTSTR audioPath, bool &readOnly)

  Opens an existing audio file on disk, automatically detecting the sequence format and opening it accordingly.
- bool Close ()

  Closes an open audio file, writing the audio header to the AUD or BWAV file and closing it.

PLAYBACK FUNCTIONS

- bool PreparePlayback ()

  Ask audio file to stop recording and prepare playback.

READ/WRITE FUNCTIONS

- bool Write (HAudioData *audioData)

  Write the specified audio in the current sequence, which will be appended at the current recording position.
- bool Read (HAudioData *audioData, unsigned __int64 index, unsigned int buffers)

  Read an audio data from the current audio file.
- bool Read (HAudioData *audioData, unsigned int buffers)

  Read an audio data at current read position from the current audio file.
- bool Read (HAudioData *audioData, long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS, unsigned int buffers=1)

  Read an audio data at specifiled time stamp from the current audio file.
- bool SetReadingPosition (unsigned __int64 position)

  Set the start position of the read.
- unsigned __int64 GetAllocatedAudioDatas ()

  Retrieve the number of audio blocks stored in the current audio file.
- unsigned __int64 GetRecordingPosition ()

  Retrieve the position where the next audio data will be written in the current audio file.
- unsigned __int64 GetReadingPosition ()

  Retrieves the current read position, by returning sequence index of the current read position.

TIMESTAMP SEARCH FUNCTIONS

- unsigned __int64 GetPositionByTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS)

  Search the audio file to find the last audio that had been captured at a given time.
- bool GetTimeStamp (unsigned __int64 index, long &timeStamp, unsigned short &timeStampMS, unsigned short &timeStampUS)

  Retrieve the timestamp of a given frame in the sequence.
- bool SetStartTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS)

  Set BWAV audio file reference timestamp at start.
- bool SetPositionByTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short time↩StampUS)

  Modifies read sample Position by specifiled time stamp.

Friends

- class **HAviFile**

3.14.1 Detailed Description

This class is used to manage the creation/reading/writing of an audio file.

Note : The audio-related classes are still in active development and should be considered as beta.

3.14.2 Member Function Documentation

3.14.2.1 void HAudioFile::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.14.2.2 eHErrorCode HAudioFile::GetLastError ( )

See also

HGrabModule::GetLastError

3.14.2.3 long HAudioFile::GetAudioSamplesPerBuffer ( )

Retrieves Samples size stored in the current audio file.

Returns

Samples size

3.14.2.4 long HAudioFile::GetAudioChannelCount ( )

Retrieves numbers of channels stored in the current audio file.

Returns

numbers of channels

3.14.2.5 eHAudioFormat HAudioFile::GetAudioFormat ( )

Retrieves audio format stored in the current audio file.

Returns

audio format

3.14.2.6 long HAudioFile::GetAudioDepth ( )

Retrieves the bit depth of the audio stored in the current audio file.

Returns

The audio bit depth.

3.14.2.7 long HAudioFile::GetAudioDepthReal ( )

Retrieves the real bit depth of the audio stored in the current audio file.

Returns

The audio bit real depth.

3.14.2.8 int HAudioFile::GetSampleRate ( )

Retrieves the sample rate stored in the current audio file.

Returns

sample rate

3.14.2.9 bool HAudioFile::GetAufioFileType (eAudioFileFormat & *format*)

Retrieves audio file format in the current audio file.

Returns

false if unknown type

Parameters

out	*format*	Audio file type.It can be aud and WBAV

3.14.2.10 bool HAudioFile::Create (eAudioFileFormat *fileType,* LPCTSTR *audioPath,* eAudFileMode *audoMode =* H_AUD_FILE_AUD_BLOCK)

Creates an empty sequence file on disk or in RAM.

It will also reset the following settings to their default values.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*fileType*	Audio file type.It can be aud and WBAV
in	*audioPath*	The full path where the audio is to be saved (ex: c:/capture.aud, c↩ :/capture.wav).
in	*audoMode*	aud file type mode reserved for future use.Should be H_AUD_FILE_AUD_B↩ LOCK

3.14.2.11 bool HAudioFile::Open (LPCTSTR *audioPath,* bool & *readOnly*)

Opens an existing audio file on disk, automatically detecting the sequence format and opening it accordingly.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*audioPath*	The full path where the sequence is located. (ex : c:/capture.aud).

out	*readOnly*	True if open as read only or Flase if open as read and write

3.14.2.12 bool HAudioFile::Close ( )

Closes an open audio file, writing the audio header to the AUD or BWAV file and closing it.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.14.2.13 bool HAudioFile::PreparePlayback ( )

Ask audio file to stop recording and prepare playback.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.14.2.14 bool HAudioFile::Write (HAudioData * *audioData*)

Write the specified audio in the current sequence, which will be appended at the current recording position.

Using SetRecordingPosition will set a specific recording position, overwriting previously recorded audio data. Once the function completes, the recording position will be set to append the next audio data to the current one.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*audioData*	The audio to be saved to the audio file.

3.14.2.15 bool HAudioFile::Read (HAudioData * *audioData*, unsigned __int64 *index*, unsigned int *buffers*)

Read an audio data from the current audio file.

An HAudioData will need to be allocated to be read and freed once it is done with.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*audioData*	An HAudioData to hold the audio data read from the audio file.

in	*index*	The requested audio block index. This ranges from 1 to the value retrieved with GetAllocatedAudioDatas().
in	*buffers*	the number of auido blocks to read

3.14.2.16 bool HAudioFile::Read (HAudioData * *audioData,* unsigned int *buffers*)

Read an audio data at current read position from the current audio file.

An HAudioData will need to be allocated to be read and freed once it is done with.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*audioData*	An HAudioData to hold the audio data read from the audio file.
in	*buffers*	the number of auido blocks to read

3.14.2.17 bool HAudioFile::Read (HAudioData * *audioData,* long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS,* unsigned int *buffers* = 1)

Read an audio data at specifiled time stamp from the current audio file.

An HAudioData will need to be allocated to be read and freed once it is done with.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*audioData*	An HAudioData to hold the audio data read from the audio file.
in	*timeStamp*	The absolute timestamp in seconds.
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.
in	*buffers*	the number of auido blocks to read

3.14.2.18 bool HAudioFile::SetReadingPosition (unsigned __int64 *position*)

Set the start position of the read.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

position	index [in] The desired playback position.

3.14.2.19 unsigned __int64 HAudioFile::GetAllocatedAudioDatas ( )

Retrieve the number of audio blocks stored in the current audio file.

Returns

The number of audio blocks in the sequence.

3.14.2.20 unsigned __int64 HAudioFile::GetRecordingPosition ( )

Retrieve the position where the next audio data will be written in the current audio file.

Returns

The current recording position.

3.14.2.21 unsigned __int64 HAudioFile::GetReadingPosition ( )

Retrieves the current read position, by returning sequence index of the current read position.

Returns

The current read position.

3.14.2.22 unsigned __int64 HAudioFile::GetPositionByTimeStamp (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Search the audio file to find the last audio that had been captured at a given time.

Returns

the audio data index.

Parameters

in	*timeStamp*	The absolute timestamp in seconds.
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.14.2.23 bool HAudioFile::GetTimeStamp (unsigned __int64 *index,* long & *timeStamp,* unsigned short & *timeStampMS,* unsigned short & *timeStampUS*)

Retrieve the timestamp of a given frame in the sequence.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The audio data index.
out	*timeStamp*	The absolute timestamp in seconds.
out	*timeStampMS*	The millisecond part of the timestamp.

out	*timeStampUS*	The microsecond part of the timestamp.

3.14.2.24 bool HAudioFile::SetStartTimeStamp (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Set BWAV audio file reference timestamp at start.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*timeStamp*	The absolute timestamp in seconds.
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.14.2.25 bool HAudioFile::SetPositionByTimeStamp (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Modifies read sample Position by specifiled time stamp.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*timeStamp*	The absolute timestamp in seconds.
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.15 HAudioPlayer Class Reference

This class can be use to control audio playback from an .AUD ,.MP4, MP3 or AAC audio file.

```
#include <HAudioPlayer.h>
```

Public Member Functions

- CHermesAudioPlayer ∗ **GetAudioPlayer** ()

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

FILE MANAGEMENT FUNCTIONS

- bool SetFileSource (HAudioFile ∗file)

  Set .AUD audio file source.

- bool SetFileSource (HMP4File *file)
  Set .MP4 audio file source.
- bool SetFileSource (HAACFile *file)
  Set .AAC audio file source.
- bool SetFileSource (HMP3File *file)
  Set .MP3 audio file source.
- bool SetLiveSource (int sampleRate, int samples, int channels, eHAudioFormat format, long audioDepth, long audioDepthReal)
  Set live audio source from a device that deliver a live audio stream.
- bool OpenPlayer (HAudioDriverDirectX *driver, int handle, HWND window, CAudioPlayerCallback *callbacks, int mapLen=0, int *maps=NULL, bool autoConfig=true)
  Prepare the file to be played through a DirectX audio device.
- bool OpenPlayer (HAudioDriverASIO *driver, int handle, CAudioPlayerCallback *callbacks, int mapLen, int *maps, bool autoConfig=true)
  Prepare the file to be played through an ASIO audio device.
- bool OpenPlayer (HGrabAudioModule *driver, CAudioPlayerCallback *callbacks, int mapLen, int *maps, bool autoConfig=true)
  Prepare the file to be played through a video grabber device with audio capabilities.
- void ClosePlayer ()
  Close the player, freeing all the allocated memory.

PLAYBACK FUNCTIONS

- bool Play ()
  Start the playback of an audio file.
- bool Stop ()
  Stop the playback of an audio file.
- bool IsPlaying ()
  Check if an audio file is currently playing.
- bool PushLiveAudioData (HAudioData *data)
  Push live audio to player for playback. audio souce must set by SetLiveSource().

BROWSING FUNCTIONS

- bool MoveTo (unsigned __int64 bufferIndex)
  Move to a specific buffer index in the audio file.

3.15.1 Detailed Description

This class can be use to control audio playback from an .AUD ,.MP4, MP3 or AAC audio file.

Note : The audio-related classes are still in active development and should be considered as beta.

3.15.2 Member Function Documentation

3.15.2.1 void HAudioPlayer::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.15.2.2 eHErrorCode HAudioPlayer::GetLastError ( )

See also

HGrabModule::GetLastError

3.15.2.3 bool HAudioPlayer::SetFileSource (HAudioFile * *file*)

Set .AUD audio file source.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The .AUD audio file to playback.

3.15.2.4 bool HAudioPlayer::SetFileSource (HMP4File * *file*)

Set .MP4 audio file source.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The .MP4 file to playback.

3.15.2.5 bool HAudioPlayer::SetFileSource (HAACFile * *file*)

Set .AAC audio file source.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The .AAC file to playback.

3.15.2.6 bool HAudioPlayer::SetFileSource (HMP3File * *file*)

Set .MP3 audio file source.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The .MP3 file to playback.

3.15.2.7 bool HAudioPlayer::SetLiveSource (int *sampleRate,* int *samples,* int *channels,* eHAudioFormat *format,* long *audioDepth,* long *audioDepthReal*)

Set live audio source from a device that deliver a live audio stream.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*sampleRate*	audio sample rate.
in	*samples*	number of audio samples per buffer.
in	*channels*	number of audio channels.
in	*format*	audio format.
in	*audioDepth*	audio bit depth.
in	*audioDepthReal*	audio bit real.

3.15.2.8 bool HAudioPlayer::OpenPlayer (HAudioDriverDirectX * *driver*, int *handle*, HWND *window*, CAudioPlayerCallback * *callbacks*, int *mapLen* = 0, int * *maps* = NULL, bool *autoConfig* = true)

Prepare the file to be played through a DirectX audio device.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*driver*	The DirectX audio driver to use.
in	*handle*	The handle to a DirectX device that is managed by the driver.
in	*window*	The handle of a valid window. The audio might be muted while the window doesn't have the focus.
in	*callbacks*	The callback class used to notify the user of internal actions.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to playback to. For example, if you want to playback on channels 1,3 & 4, the maps array would be [1,3,4].
in	*autoConfig*	If true, the audio driver output format will be set to the same format as the file. Else, the output format must be set manually from the HAudioDriverDirectX class.);

3.15.2.9 bool HAudioPlayer::OpenPlayer (HAudioDriverASIO * *driver*, int *handle*, CAudioPlayerCallback * *callbacks*, int *mapLen*, int * *maps*, bool *autoConfig* = true)

Prepare the file to be played through an ASIO audio device.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*driver*	The ASIO driver to use.
in	*handle*	The handle to an ASIO device that is managed by the driver.
in	*callbacks*	The callback class used to notify the user of internal actions.

in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to playback to. For example, if you want to playback on channels 1,3 & 4, the maps array would be [1,3,4].
in	*autoConfig*	If true, the audio driver output format will be set to the same format as the file. Else, the output format must be set manually from the HAudioDriverASIO class.

3.15.2.10 bool HAudioPlayer::OpenPlayer (HGrabAudioModule * *driver*, CAudioPlayerCallback * *callbacks*, int *mapLen*, int * *maps*, bool *autoConfig* = `true`)

Prepare the file to be played through a video grabber device with audio capabilities.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*driver*	The video grabber device with audio capabilities to use.
in	*callbacks*	The callback class used to notify the user of internal actions.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to playback to. For example, if you want to playback on channels 1,3 & 4, the maps array would be [1,3,4].
in	*autoConfig*	If true, the audio driver output format will be set to the same format as the file. Else, the output format must be set manually from the HGrabAudioModule class.

3.15.2.11 bool HAudioPlayer::Play ( )

Start the playback of an audio file.

OpenPlayer(...) must have been called previously.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.15.2.12 bool HAudioPlayer::Stop ( )

Stop the playback of an audio file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.15.2.13 bool HAudioPlayer::IsPlaying ( )

Check if an audio file is currently playing.

Returns

True if currently in playback

3.15.2.14 bool HAudioPlayer::PushLiveAudioData (HAudioData * *data*)

Push live audio to player for playback. audio souce must set by SetLiveSource().

Parameters

in	*data*	source audio data.

3.15.2.15 bool HAudioPlayer::MoveTo (unsigned __int64 *bufferIndex*)

Move to a specific buffer index in the audio file.

This function won't work if called while a playback is running. Typically, call MoveTo, then Play.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*bufferIndex*	The index of the buffer to move to.

3.16 HAudioRecorder Class Reference

Control audio recording from an audio device to an audio file.

```
#include <HAudioRecorder.h>
```

Public Member Functions

- HAudioRecorder (void)

  Constructor.
- virtual ~HAudioRecorder (void)

  Destructor.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

CALLBACK FUNCTIONS

- void SetCallback (CAudioRecorderCallback *callback)

  Set the callback call used by the audio recorder.

RECORDING FUNCTIONS

- bool SetAudioDriver (HAudioDriverDirectX *driver, int handle)

  Set the DirectX compatible audio device that will be used for audio capture.
- bool SetAudioDriver (HAudioDriverASIO *driver, int handle, int mapLen, int *maps, int sampleRate)

  Set the ASIO compatible audio device that will be used for audio capture.
- bool SetAudioDriver (HGrabAudioModule *driver)

  Set the video capture device with audio capabilities that will be used for audio capture.
- bool RemoveAudioDriver ()

  Terminate the association between the audio recorder and the current audio driver.
- bool SetAudioFile (HAudioFile *file)

  Set the target .AUD audio file for the acquired audio buffered.
- bool SetAudioFile (HAACFile *file, int sampleRate, int bitRate, int mapLen, int *maps)

Set the target .AAC audio file for the acquired audio buffered.

- bool SetAudioFile (HMP3File *file, int sampleRate, int bitRate, int mapLen, int *maps)

  Set the target .MP3 audio file for the acquired audio buffered.
- bool SetAudioFile (HMP4File *file, int sampleRate, int bitRate, int mapLen, int *maps)

  Set the target MP4 audio file for the acquired audio buffered.
- bool SetAudioFile (HMPEGTsFile *file, int sampleRate, int bitRate, int mapLen, int *maps)

  Set the target .TS audio file for the acquired audio buffered.
- bool OpenChannels ()

  Open the audio channels.
- bool CloseChannels ()

  Close the audio channels.
- bool AreChannelsOpen ()

  Check if the audio channels are currently open.
- bool StartRecording ()

  Start streaming the audio buffers received from the audio device to the target audio file.
- bool StopRecording ()

  Stop audio recording to the audio file.
- bool IsRecording ()

  Check if the audio recorder is currently recording.
- double GetAudioInputLevel ()

  Retrieve the highest input level across all channels.
- double GetAudioInputLevel (int ch)

  Retrieve the input level of a channel.

PASSTHROUGH FUNCTIONS

- void EnablePassthrough (HAudioPlayer *player)

  Open the audio pass-through.
- void DisablePassthrough ()

  Close the audio pass-through.

BUFFERS FUNCTIONS

- void SetBufferCount (unsigned int count)

  Set the number of buffers to be used by the audio capture mechanism.
- unsigned int GetBufferCount ()

  Get the number of buffers used by the grabbing mechanism.
- long GetBufferUsage ()

  Get the number of frames waiting to be processed by the OnDataReceived callback.
- long GetMaxBufferUsage ()

  Get the maximum number of buffers reached since acquisition started (since StartStreaming()).

3.16.1 Detailed Description

Control audio recording from an audio device to an audio file.

Supporting: .AUD, .MP3, .TS, .MP4, .AAC file format.

3.16.2 Member Function Documentation

3.16.2.1 void HAudioRecorder::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.16.2.2 eHErrorCode HAudioRecorder::GetLastError ( )

See also

HGrabModule::GetLastError

3.16.2.3 void HAudioRecorder::SetCallback (CAudioRecorderCallback * *callback*)

Set the callback call used by the audio recorder.

Parameters

in	*callback*	A pointer to a CAudioRecorderCallback.

3.16.2.4 bool HAudioRecorder::SetAudioDriver (HAudioDriverDirectX * *driver*, int *handle*)

Set the DirectX compatible audio device that will be used for audio capture.

The device must have been initialized. The DirectX driver only allows single or dual channels acquisition (i.e. mono, stereo)

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*driver*	The DirectX driver to use.
in	*handle*	The handle to a DirectX device that is managed by the driver.

3.16.2.5 bool HAudioRecorder::SetAudioDriver (HAudioDriverASIO * *driver*, int *handle*, int *mapLen*, int * *maps*, int *sampleRate*)

Set the ASIO compatible audio device that will be used for audio capture.

The device must have been initialized.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*driver*	The ASIO driver to use.
in	*handle*	The handle to an ASIO device that is managed by the driver.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to capture from. For example, if you want to capture from channels 1,3 & 4, the maps array would be [1,3,4].
in	*sampleRate*	The acquisition rate (in samples per seconds)

3.16.2.6 bool HAudioRecorder::SetAudioDriver (HGrabAudioModule * *driver*)

Set the video capture device with audio capabilities that will be used for audio capture.

The device must have been initialized.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*driver*	The video capture device with audio capabilities to use.

3.16.2.7 bool HAudioRecorder::SetAudioFile (HAudioFile * *file*)

Set the target .AUD audio file for the acquired audio buffered.

The target file must be already open.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The target audio file.

3.16.2.8 bool HAudioRecorder::SetAudioFile (HAACFile * *file*, int *sampleRate*, int *bitRate*, int *mapLen*, int * *maps*)

Set the target .AAC audio file for the acquired audio buffered.

The target file must already be open.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The target .AAC audio file.
in	*sampleRate*	Audio sample rate. Should match the audio capture device settings.
in	*bitRate*	AAC compresion bit rate in bit per second.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to write.

3.16.2.9 bool HAudioRecorder::SetAudioFile (HMP3File * *file*, int *sampleRate*, int *bitRate*, int *mapLen*, int * *maps*)

Set the target .MP3 audio file for the acquired audio buffered.

The target file must already be open.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The target .MP3 audio file.
in	*sampleRate*	Audio sample rate. Should match the audio capture device settings.
in	*bitRate*	MP3 compresion bit rate in bit per second.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to write.

3.16.2.10 bool HAudioRecorder::SetAudioFile (HMP4File * *file*, int *sampleRate*, int *bitRate*, int *mapLen*, int * *maps*)

Set the target MP4 audio file for the acquired audio buffered.

The target file must already be open.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The target .MP4 audio file.
in	*sampleRate*	Audio sample rate. Should match the audio capture device settings.
in	*bitRate*	AAC compresion bit rate in bit per second.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to write.

3.16.2.11 bool HAudioRecorder::SetAudioFile (HMPEGTsFile * *file*, int *sampleRate*, int *bitRate*, int *mapLen*, int * *maps*)

Set the target .TS audio file for the acquired audio buffered.

The target file must already be open.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*file*	The target .TS audio file.
in	*sampleRate*	Audio sample rate. Should match the audio capture device settings.
in	*bitRate*	AAC compresion bit rate in bit per second.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to write.

3.16.2.12 bool HAudioRecorder::OpenChannels ( )

Open the audio channels.

Starts listening to the audio device. SetAudioDriver(...) must be called prior to this function. This doesn't start the recording itself, but it allows you to hear the audio input.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.16.2.13 bool HAudioRecorder::CloseChannels ( )

Close the audio channels.

This will stop the audio device from acquiring anymore audio buffers.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.16.2.14 bool HAudioRecorder::AreChannelsOpen ( )

Check if the audio channels are currently open.

Returns

True if the channels are open, false otherwise.

3.16.2.15 bool HAudioRecorder::StartRecording ( )

Start streaming the audio buffers received from the audio device to the target audio file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.16.2.16 bool HAudioRecorder::StopRecording ( )

Stop audio recording to the audio file.

The audio channels still remain open.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.16.2.17 bool HAudioRecorder::IsRecording ( )

Check if the audio recorder is currently recording.

Returns

True if recording, false otherwise.

3.16.2.18 double HAudioRecorder::GetAudioInputLevel ( )

Retrieve the highest input level across all channels.

Returns

The highest input level across all open channels

3.16.2.19 double HAudioRecorder::GetAudioInputLevel (int *ch*)

Retrieve the input level of a channel.

Returns

The input level on that channel.

Parameters

ch	The channel

3.16.2.20 void HAudioRecorder::EnablePassthrough (HAudioPlayer * *player*)

Open the audio pass-through.

If the pass-through is open, when the channels are open, the device will also send audio buffer to the default audio-out. Note : On Windows 7 and later, pass-through is always enabled.

Parameters

in	*player*	The target audio player.

3.16.2.21 void HAudioRecorder::SetBufferCount (unsigned int *count*)

Set the number of buffers to be used by the audio capture mechanism.

Having a large number of buffers will help to prevent audio drops. However, having too many buffers will quickly eat the available RAM, so there is a compromise to be made. The default 5 buffers should be enough for most applications. SetBufferCount must be called before OpenChannels, as the buffer pool is created in the Load function.

Parameters

in	*count*	The number of buffers to create.

3.16.2.22 unsigned int HAudioRecorder::GetBufferCount ( )

Get the number of buffers used by the grabbing mechanism.

The default value is 5, but can be changed with the SetBufferCount function.

Returns

The number of buffers used by the audio capture mechanism.

3.16.2.23 long HAudioRecorder::GetBufferUsage ( )

Get the number of frames waiting to be processed by the OnDataReceived callback.

If this number reaches the total buffer count, the application will start dropping data. The best place to call this function is in the OnDataReceived callback function.

Returns

The number of buffers waiting to be processed.

3.16.2.24 long HAudioRecorder::GetMaxBufferUsage ( )

Get the maximum number of buffers reached since acquisition started (since StartStreaming()).

Returns

The maximum buffer usage since acquisition started.

3.17 HAutoLocker Class Reference

The HAutoLocker is a mutex class used through the API.

```
#include <HLocker.h>
```

Public Member Functions

- HAutoLocker (CRITICAL_SECTION *section)

  Enter the critical section.
- ~HAutoLocker ()

  Leave the critical section.

3.17.1 Detailed Description

The HAutoLocker is a mutex class used through the API.

3.18 HAviExporter Class Reference

Export a sequence file to an AVI.

```
#include <HAviExporter.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

SETTINGS FUNCTIONS

- bool GetVideoCodecList (int *count, HItem **items)

  Get a list of every video codec found on the system.
- bool SetVideoCodec (unsigned int ID)

  Select the video codec to be used for encoding.
- bool ShowVideoCodecSettings ()

  Show the settings dialog for the currently selected video codec.
- bool GetAudioSourceList (int *count, HItem **items)

  Get a list of every audio source found on the system.
- bool SetAudioSource (unsigned int ID)

  Select the audio source to be used for recording.
- bool GetAudioCodecList (int *count, HItem **items)

  Get a list of every audio codec found on the system.
- bool SetAudioCodec (unsigned int ID)

  Select the audio codec to be used for encoding.

- bool ShowAudioCodecSettings ()
  Show the settings dialog for the currently selected audio codec.
- bool UseSpecialCodec (eHVideoCodec codec, int sizeOfParam, void *codecParam)
  Use sepcified codec,whichsupport by Norpix.
- bool ForceFrameRate (double framerate)
  Force the AVI frame rate to a specific value.

EXPORT FUNCTIONS

- bool BeginExport (LPCTSTR filename, double framerate)
  Initiate the export process.
- bool ExportImage (HImage *image)
  Append an image to the AVI file.
- bool EndExport ()
  Completes the export process.
- bool IsExporting ()
  Check if an export is currently in progress.

3.18.1 Detailed Description

Export a sequence file to an AVI.

(Obsolete, use HAviFile instead) CoInitialize(NULL) must be called on the same thread that will be using this object. Once you have finished with it, call CoUninitialize().

3.18.2 Member Function Documentation

3.18.2.1 void HAviExporter::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.18.2.2 eHErrorCode HAviExporter::GetLastError ( )

See also

HGrabModule::GetLastError

3.18.2.3 bool HAviExporter::GetVideoCodecList (int * *count*, HItem ** *items*)

Get a list of every video codec found on the system.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*count*	The number of codecs.

out	*items*	An array of items, one for each codec.

3.18.2.4 bool HAviExporter::SetVideoCodec (unsigned int *ID*)

Select the video codec to be used for encoding.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*ID*	ID of the video codec. Get it from one of the items retreived from GetVideo↩ CodecList().

3.18.2.5 bool HAviExporter::ShowVideoCodecSettings ( )

Show the settings dialog for the currently selected video codec.

Some video codecs don't have settings dialog in which case this function will have no effect.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.18.2.6 bool HAviExporter::GetAudioSourceList (int * *count*, HItem ** *items*)

Get a list of every audio source found on the system.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*count*	The number of audio input.
out	*items*	An array of items, one for each audio input.

3.18.2.7 bool HAviExporter::SetAudioSource (unsigned int *ID*)

Select the audio source to be used for recording.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*ID*	ID of the audio source. Get it from one of the items retreived from GetAudio↩ SourceList().

3.18.2.8 bool HAviExporter::GetAudioCodecList (int * *count,* HItem ** *items*)

Get a list of every audio codec found on the system.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*count*	The number of codecs.
out	*items*	An array of items, one for each codec.

3.18.2.9 bool HAviExporter::SetAudioCodec (unsigned int *ID*)

Select the audio codec to be used for encoding.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*ID*	ID of the audio codec. Get it from one of the items retreived from GetAudio↩ CodecList().

3.18.2.10 bool HAviExporter::ShowAudioCodecSettings ( )

Show the settings dialog for the currently selected audio codec.

Some audio codecs don't have settings dialog in which case this function will have no effect.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.18.2.11 bool HAviExporter::UseSpecialCodec (eHVideoCodec *codec,* int *sizeOfParam,* void * *codecParam*)

Use sepcified codec,whichsupport by Norpix.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*codec*	Codec type
in	*sizeOfParam*	Codec parameter buffer size
in	*codecParam*	Codec parameter buffer

3.18.2.12 bool HAviExporter::ForceFrameRate (double *framerate*)

Force the AVI frame rate to a specific value.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*framerate*	The frame rate in frames per second.

3.18.2.13 bool HAviExporter::BeginExport (LPCTSTR *filename,* double *framerate*)

Initiate the export process.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The file path, ex : "C:\folder\file.avi"
in	*framerate*	The default frame rate of the AVI.

3.18.2.14 bool HAviExporter::ExportImage (HImage * *image*)

Append an image to the AVI file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The image to export.

3.18.2.15 bool HAviExporter::EndExport ( )

Completes the export process.

Writes the header and close the AVI file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.18.2.16 bool HAviExporter::IsExporting ( )

Check if an export is currently in progress.

An export starts with a call to BeginExport and ends with a call to EndExport.

Returns

True if an export is in progress.

3.19 HAviFile Class Reference

This class manages AVI recording & playback.

```
#include <HAviFile.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

CALLBACKS RELATED FUNCTIONS

- bool SetCallbacks (HAviFileCallbacks *userCallbacks, unsigned int userData=0, void *userPtr=NULL)

  Set the callback class used by the AVIFile module.

FILE MANAGEMENT AND RECORDING FUNCTIONS

- bool CreateAviFile (LPCTSTR filePath, double framerate)

  CreateAviFile will create a new AVI file and open it in recording mode.
- bool CreateAviFile (LPCTSTR filePath, double framerate, HAudioFile *audioFile, int leftChannelOut, int rightChannelOut, long startTime=0, unsigned short startTimeMS=0, unsigned short startTimeUS=0, long stopTime=0, unsigned short stopTimeMS=0, unsigned short stopTimeUS=0, long adjustRefTime=0, unsigned short adjustRefTimeMS=0, unsigned short adjustRefTimeUS=0)

  Export to AVI with audio from an audio file.
- bool CreateAviFile (LPCTSTR filePath, double framerate, HAACFile *audioFile, int leftChannelOut, int rightChannelOut, long startTime=0, unsigned short startTimeMS=0, unsigned short startTimeUS=0, long stopTime=0, unsigned short stopTimeMS=0, unsigned short stopTimeUS=0, long adjustRefTime=0, unsigned short adjustRefTimeMS=0, unsigned short adjustRefTimeUS=0)

  Export to AVI with audio from an audio file.
- bool CreateAviFile (LPCTSTR filePath, double framerate, HMP3File *audioFile, int leftChannelOut, int rightChannelOut, long startTime=0, unsigned short startTimeMS=0, unsigned short startTimeUS=0, long stopTime=0, unsigned short stopTimeMS=0, unsigned short stopTimeUS=0, long adjustRefTime=0, unsigned short adjustRefTimeMS=0, unsigned short adjustRefTimeUS=0)

  Export to AVI with audio from an audio file.
- bool OpenAviFile (LPCTSTR filePath)

  Open an existing AVI file in playback mode.
- bool AddFrame (HImage *image)

  Add an image at the end of an AVI opened in recording mode.
- bool StopRecording ()

  Stop the video and audio acquisition of the current AVI file.
- bool CloseAviFile ()

  Close the current AVI file.
- eHMovieMode GetAviMode ()

  Get the current AVI mode (playback or recording).

VIDEO CODEC FUNCTIONS

- int GetVideoCodecCount ()

  Get the number of available video codecs.
- LPCTSTR GetVideoCodecName (int index)

  Get the name of each video codec.
- bool SetVideoCodec (int index)

  Select the video codec that will be used for compressing the images.
- bool ShowVideoCodecSettings ()

  Display a 3rd party dialog allowing to change the currently selected video codec settings.

AUDIO SOURCE & AUDIO CODEC FUNCTIONS

- int GetAudioSourceCount ()

  Get the number of available audio sources.
- LPCTSTR GetAudioSourceName (int index)

  Get the name of each audio source.
- bool SetAudioSource (int index)

  Select the audio source that will be used for audio capture.
- bool **ShowVideoSourceSettings** ()
- int GetAudioCodecCount ()

  Get the number of available audio codecs.
- LPCTSTR GetAudioCodecName (int index)

  Get the name of each audio codec.
- bool SetAudioCodec (int index)

  Select the audio codec that will be used for compressing the audio.
- bool ShowAudioCodecSettings ()

  Display a 3rd party dialog allowing to change the currently selected audio codec settings.

CODEC PARAMETERS FUNCTIONS

- eHVideoCodec DetectVideoCodecType (int index)

  Detect if a specific video codec can take additionnal parameters.
- void ResetCodecParams ()

  Reset the parameters used for each codec.
- bool SetCineFormCodecRawParams (CCineFormRawParam *params)

  Set the additionnal parameters if the video codec is of type H_CODEC_CINEFORM_RAM.
- bool SetCineFormCodecHDParams (CCineFormHDParam *params)

  Set the additionnal parameters is the video codec is of type H_CODEC_CINEFORM_HD.

RECORDING SETUP FUNCTION

- bool ForceFrameRate (double framerate)

  Ignore the rate at which images are capture and force a specified framerate to the file header.
- void **ForceBGRConversion** (bool enable)
- bool IsLiveRecording ()

  Check if the AVI is being recorded in "live" mode.
- void SetLiveRecording (bool enable)

  Enable or disable the live recording mode.

PLAYBACK FUNCTIONS

- bool Play ()

  Start the playback of an AVI file.
- bool Stop ()

  Stop the playback of an AVI file.
- bool Pause ()

  Pause/resume the playback of an AVI file.
- bool IsPlaying ()

Check if the AVI is currently playing.

- bool IsPaused ()

  Check if the AVI playback is paused.
- bool IsStopped ()

  Check if the AVI playback is stopped.
- bool ReadImage (HImage *image, unsigned int index)

  Retrieve the image at index.
- bool GetIndexAt (long time, unsigned short timeMS, unsigned short timeUS, unsigned int &index)

  Retrieve the index at the specified time stamp.
- void SetLooping (bool enable)

  Enable or disable looping while doing playback.
- bool IsLooping ()

  Check if the AVI loops when at the end of playback.
- unsigned long GetFrameCount ()

  Retrieve the number of images in the AVI file.
- unsigned long GetCurrentPosition ()

  Get the index of the current frame while in playback mode.
- double GetPlaybackFrameRate ()

  Get the frame rate of the AVI file.
- eHAviPlaybackFormat GetPlaybackFormat ()

  Retrieve the current image format used for playback.
- void SetPlaybackFormat (eHAviPlaybackFormat format)

  Set the image format used during playback.
- double GetPlaybackRate ()

  Retrieve the current AVI playback rate in percentage of the original.
- void SetPlaybackRate (double rate)

  Set the AVI playback rate.

METADATA SUPPORT (RIFF INFO Tags)

- bool SetTitle (LPCTSTR title)

  Set AVI title infomation.
- bool GetTitle (LPTSTR title, UINT *length)

  Get AVI title infomation.
- bool SetCopyright (LPCTSTR copyright)

  Set copyright infomation.
- bool GetCopyright (LPTSTR copyright, UINT *length)

  Get copyright infomation.
- bool SetCreationDateTime (LPCTSTR datetime)

  Set AVI create time.
- bool GetCreationDateTime (LPTSTR datetime, UINT *length)

  Set AVI create time.
- bool SetDateTimeOriginal (LPCTSTR datetime)

  Set AVI orignal create date time.
- bool GetDateTimeOriginal (LPTSTR datetime, UINT *length)

  Get AVI orignal create date time.
- bool SetAuthor (LPCTSTR author)

  Set AVI Author infomation.
- bool GetAuthor (LPTSTR author, UINT *length)

  Get AVI Author infomation.
- bool SetSoftwareName (LPCTSTR name)

  Set software name.
- bool GetSoftwareName (LPTSTR name, UINT *length)

  Get software name.

3.19.1 Detailed Description

This class manages AVI recording & playback.

This class takes one or more images in HImage format and saves them to an AVI file on disk. Any installed compression codec can be used.

The list of codecs installed on the computer can be found in Windows Media Player. Open the Help menu and select 'About Windows Media Player'. Finally, click on 'Technical Support Information'. At the bottom of the page, you'll find a list of the video and audio codecs installed on your system.

This class can also playback any AVI file recorded by it. An AVI file can either be used for recording or for playback (not at the same time).

For the recording, the call order is :

- Call CreateAviFile
- Use the recording setup functions if you want to override the default settings
- Call AddFrame for every frame
- Call StopRecording

You can only use the recording functions if CreateAviFile was called. You can only use the playback functions if OpenAviFile was called.

3.19.2 Member Function Documentation

3.19.2.1 void HAviFile::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.19.2.2 eHErrorCode HAviFile::GetLastError ( )

See also

HGrabModule::GetLastError

3.19.2.3 bool HAviFile::SetCallbacks (HAviFileCallbacks * *userCallbacks,* unsigned int *userData* = 0, void * *userPtr* = NULL)

Set the callback class used by the AVIFile module.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*userCallbacks*	The callback class to use.

in	*userData*	User specified data that is returned by the callback functions. (optional)
in	*userPtr*	User specified pointer that is returned by the callback functions. (optional)

3.19.2.4 bool HAviFile::CreateAviFile (LPCTSTR *filePath,* double *framerate*)

CreateAviFile will create a new AVI file and open it in recording mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filePath*	The full path name of the AVI file to create. (Ex: "C:/folder/movie.avi")
in	*framerate*	The default average frame rate of the output AVI. (in frames/second), etc...

3.19.2.5 bool HAviFile::CreateAviFile (LPCTSTR *filePath,* double *framerate,* HAudioFile * *audioFile,* int *leftChannelOut,* int *rightChannelOut,* long *startTime* = 0, unsigned short *startTimeMS* = 0, unsigned short *startTimeUS* = 0, long *stopTime* = 0, unsigned short *stopTimeMS* = 0, unsigned short *stopTimeUS* = 0, long *adjustRefTime* = 0, unsigned short *adjustRefTimeMS* = 0, unsigned short *adjustRefTimeUS* = 0)

Export to AVI with audio from an audio file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filePath*	The full path name of the AVI file to create. (Ex: "C:/folder/movie.avi")
in	*framerate*	The default average frame rate of the output AVI. (in frames/second), etc...
in	*audioFile*	Audio file to export
in	*leftChannelOut*	chanel index for left channel
in	*rightChannelOut*	chanel index for right channel
in	*startTime*	Start export absolute timestamp in seconds in audio file
in	*startTimeMS*	Start export millisecond part in audio file
in	*startTimeUS*	Start export microsecond part in audio file
in	*stopTime*	stop export absolute timestamp in seconds in audio file
in	*stopTimeMS*	stop export millisecond part in audio file
in	*stopTimeUS*	stop export microsecond part in audio file
in	*adjustRefTime*	Different audio and video timestamp time
in	*adjustRefTime↩ MS*	Different audio and video millisecond part time
in	*adjustRefTime↩ US*	Different audio and video microsecond part time

3.19.2.6 bool HAviFile::CreateAviFile (LPCTSTR *filePath,* double *framerate,* HAACFile * *audioFile,* int *leftChannelOut,* int *rightChannelOut,* long *startTime* = 0, unsigned short *startTimeMS* = 0, unsigned short *startTimeUS* = 0, long *stopTime* = 0, unsigned short *stopTimeMS* = 0, unsigned short *stopTimeUS* = 0, long *adjustRefTime* = 0, unsigned short *adjustRefTimeMS* = 0, unsigned short *adjustRefTimeUS* = 0)

Export to AVI with audio from an audio file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filePath*	The full path name of the AVI file to create. (Ex: "C:/folder/movie.avi")
in	*framerate*	The default average frame rate of the output AVI. (in frames/second), etc...
in	*audioFile*	Audio file to export
in	*leftChannelOut*	chanel index for left channel
in	*rightChannelOut*	chanel index for right channel
in	*startTime*	Start export absolute timestamp in seconds in audio file
in	*startTimeMS*	Start export millisecond part in audio file
in	*startTimeUS*	Start export microsecond part in audio file
in	*stopTime*	stop export absolute timestamp in seconds in audio file
in	*stopTimeMS*	stop export millisecond part in audio file
in	*stopTimeUS*	stop export microsecond part in audio file
in	*adjustRefTime*	Different audio and video timestamp time
in	*adjustRefTime↩ MS*	Different audio and video millisecond part time
in	*adjustRefTime↩ US*	Different audio and video microsecond part time

3.19.2.7 bool HAviFile::CreateAviFile (LPCTSTR *filePath*, double *framerate*, HMP3File * *audioFile*, int *leftChannelOut*, int *rightChannelOut*, long *startTime* = 0, unsigned short *startTimeMS* = 0, unsigned short *startTimeUS* = 0, long *stopTime* = 0, unsigned short *stopTimeMS* = 0, unsigned short *stopTimeUS* = 0, long *adjustRefTime* = 0, unsigned short *adjustRefTimeMS* = 0, unsigned short *adjustRefTimeUS* = 0)

Export to AVI with audio from an audio file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filePath*	The full path name of the AVI file to create. (Ex: "C:/folder/movie.avi")
in	*framerate*	The default average frame rate of the output AVI. (in frames/second), etc...
in	*audioFile*	Audio file to export
in	*leftChannelOut*	chanel index for left channel
in	*rightChannelOut*	chanel index for right channel
in	*startTime*	Start export absolute timestamp in seconds in audio file
in	*startTimeMS*	Start export millisecond part in audio file
in	*startTimeUS*	Start export microsecond part in audio file
in	*stopTime*	stop export absolute timestamp in seconds in audio file
in	*stopTimeMS*	stop export millisecond part in audio file
in	*stopTimeUS*	stop export microsecond part in audio file
in	*adjustRefTime*	Different audio and video timestamp time
in	*adjustRefTime↩ MS*	Different audio and video millisecond part time

in	*adjustRefTime↩ US*	Different audio and video microsecond part time

3.19.2.8 bool HAviFile::OpenAviFile (LPCTSTR *filePath*)

Open an existing AVI file in playback mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filePath*	The full path name of the AVI file to open. (Ex: "C:/folder/movie.avi")

3.19.2.9 bool HAviFile::AddFrame (HImage * *image*)

Add an image at the end of an AVI opened in recording mode.

The first call to AddFrame starts the recording.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The image to save.

3.19.2.10 bool HAviFile::StopRecording ( )

Stop the video and audio acquisition of the current AVI file.

Call this when you don't have any more video frame to send and you want to stop the audio capture. Once the recording starts (after having called 'AddFrame' for the first time), the audio will be continuously captured until 'StopRecording' or 'CloseAviFile' are called. This does not apply if no audio source was selected.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.19.2.11 bool HAviFile::CloseAviFile ( )

Close the current AVI file.

Call this when you are done with the recording/playback.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.19.2.12 eHMovieMode HAviFile::GetAviMode ( )

Get the current AVI mode (playback or recording).

This is useful as some functions can only be called when the AVI file is in recording mode while some other can only be called while in playback mode.

Returns

The current mode.

3.19.2.13 int HAviFile::GetVideoCodecCount ( )

Get the number of available video codecs.

Returns

The number of available video codecs.

3.19.2.14 LPCTSTR HAviFile::GetVideoCodecName (int *index*)

Get the name of each video codec.

Returns

The name of the specified codec.

Parameters

in	*index*	The index of the video codec for which the name is retrieved. Valid range is [0, GetVideoCodecCount()-1].

3.19.2.15 bool HAviFile::SetVideoCodec (int *index*)

Select the video codec that will be used for compressing the images.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The index of the video codec to use.

3.19.2.16 bool HAviFile::ShowVideoCodecSettings ( )

Display a 3rd party dialog allowing to change the currently selected video codec settings.

The availability of this dialog varies from codec to codec.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.19.2.17 int HAviFile::GetAudioSourceCount ( )

Get the number of available audio sources.

Returns

The number of audio sources available.

3.19.2.18 LPCTSTR HAviFile::GetAudioSourceName (int *index*)

Get the name of each audio source.

Returns

The name of the specified source.

Parameters

in	*index*	The index of the audio source for which the name is retrieved. Valid range is [0, GetAudioSourceCount()-1].

3.19.2.19 bool HAviFile::SetAudioSource (int *index*)

Select the audio source that will be used for audio capture.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The index of the audio source to use.

3.19.2.20 int HAviFile::GetAudioCodecCount ( )

Get the number of available audio codecs.

Returns

The number of audio codecs available.

3.19.2.21 LPCTSTR HAviFile::GetAudioCodecName (int *index*)

Get the name of each audio codec.

Returns

The name of the specified codec.

Parameters

in	*index*	The index of the audio codec for which the name is retrieved. Valid range is [0, GetAudioCodecCount()-1].

3.19.2.22 bool HAviFile::SetAudioCodec (int *index*)

Select the audio codec that will be used for compressing the audio.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The index of the audio codec.

3.19.2.23 bool HAviFile::ShowAudioCodecSettings ( )

Display a 3rd party dialog allowing to change the currently selected audio codec settings.

The availability of this dialog varies from codec to codec.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.19.2.24 eHVideoCodec HAviFile::DetectVideoCodecType (int *index*)

Detect if a specific video codec can take additionnal parameters.

Returns

The video codec type. If the return value is H_CODEC_UNKNOWN, the codec doesn't offer additionnal parameters.

Parameters

index	index [in] The index of the video codec to check.

3.19.2.25 bool HAviFile::SetCineFormCodecRawParams (CCineFormRawParam * *params*)

Set the additionnal parameters if the video codec is of type H_CODEC_CINEFORM_RAM.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*params*	The additionnal parameters.

3.19.2.26 bool HAviFile::SetCineFormCodecHDParams (CCineFormHDParam * *params*)

Set the additionnal parameters is the video codec is of type H_CODEC_CINEFORM_HD.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*params*	The additionnal parameters.

3.19.2.27 bool HAviFile::ForceFrameRate (double *framerate*)

Ignore the rate at which images are capture and force a specified framerate to the file header.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*framerate*	The desired framerate.

3.19.2.28 bool HAviFile::IsLiveRecording ( )

Check if the AVI is being recorded in "live" mode.

Returns

True if the AVI recording is done "Live", false otherwise.

3.19.2.29 void HAviFile::SetLiveRecording (bool *enable*)

Enable or disable the live recording mode.

Unless the live recording mode is enabled, the AVI file framerate will be the one provided to the "CreateAviFile" function. If the live recording mode is enabled, the average frame rate will be calculated when the recording ends. The formula is the following : framerate = (Time of 'StopRecording' - Time of the first 'AddFrame') / Number of frames captured. So the length of the AVI file is read as the time lapse between the time at which the first frame was received via the "AddFrame" function and the time at which StopRecording was called.

Parameters

enable	True to enable live recording. By default, Live Recording is disabled.

3.19.2.30 bool HAviFile::Play ( )

Start the playback of an AVI file.

Only available while in playback mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.19.2.31 bool HAviFile::Stop ( )

Stop the playback of an AVI file.

Only available while in playback mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.19.2.32 bool HAviFile::Pause ( )

Pause/resume the playback of an AVI file.

Only available while in playback mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.19.2.33 bool HAviFile::IsPlaying ( )

Check if the AVI is currently playing.

Only available while in playback mode.

Returns

True if the AVI is playing, false in all other cases (paused, stopped, etc)

3.19.2.34 bool HAviFile::IsPaused ( )

Check if the AVI playback is paused.

Returns

True if paused, false otherwise.

3.19.2.35 bool HAviFile::IsStopped ( )

Check if the AVI playback is stopped.

Returns

True if stopped, false otherwise.

3.19.2.36 bool HAviFile::ReadImage (HImage * *image*, unsigned int *index*)

Retrieve the image at index.

Only available while in playback mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*image*	The image read from the AVI. It is the caller responsability to free the image when done with it.
in	*index*	The index of the image to read. Valid range is [0, GetFrameCount()-1].

3.19.2.37 bool HAviFile::GetIndexAt (long *time*, unsigned short *timeMS*, unsigned short *timeUS*, unsigned int & *index*)

Retrieve the index at the specified time stamp.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*time*	The time in time_t format.
in	*timeMS*	The milliseconds part.
in	*timeUS*	The microseconds part.
out	*index*	The frame index.

3.19.2.38 void HAviFile::SetLooping (bool *enable*)

Enable or disable looping while doing playback.

Parameters

in	*enable*	True to loop at the beginning of the AVI when its end is reached, False to stop.

3.19.2.39 bool HAviFile::IsLooping ( )

Check if the AVI loops when at the end of playback.

Returns

True if looping, false otherwise.

3.19.2.40 unsigned long HAviFile::GetFrameCount ( )

Retrieve the number of images in the AVI file.

Returns

The frame count.

3.19.2.41 unsigned long HAviFile::GetCurrentPosition ( )

Get the index of the current frame while in playback mode.

Returns

The current index.

3.19.2.42 double HAviFile::GetPlaybackFrameRate ( )

Get the frame rate of the AVI file.

Returns

The frame rate in frames per second.

3.19.2.43 eHAviPlaybackFormat HAviFile::GetPlaybackFormat ( )

Retrieve the current image format used for playback.

Returns

The playback format.

3.19.2.44 void HAviFile::SetPlaybackFormat (eHAviPlaybackFormat *format*)

Set the image format used during playback.

By default, the native format is used but calling this function can force the image to BGR or BGRx.

Parameters

format	format [in] The format to use

3.19.2.45 double HAviFile::GetPlaybackRate ( )

Retrieve the current AVI playback rate in percentage of the original.

1.0 means regular speed, 0.5 is half-speed, 2.0 will play twice as fast and so on.

Returns

The playback rate (in frames per second).

3.19.2.46 void HAviFile::SetPlaybackRate (double *rate*)

Set the AVI playback rate.

Default is 1.0, 0.5 is half-speed, 2.0 will play twice as fast and so on. If you change the playback rate during playback, the HAviFile will automaticaly pause the current playback, set the new rate and then resume the playback.

Parameters

in	*rate*	The playback rate in percentage.

3.19.2.47 bool HAviFile::SetTitle (LPCTSTR *title*)

Set AVI title infomation.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.48 bool HAviFile::GetTitle (LPTSTR *title,* UINT * *length*)

Get AVI title infomation.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.49 bool HAviFile::SetCopyright (LPCTSTR *copyright*)

Set copyright infomation.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.50 bool HAviFile::GetCopyright (LPTSTR *copyright,* UINT * *length*)

Get copyright infomation.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.51 bool HAviFile::SetCreationDateTime (LPCTSTR *datetime*)

Set AVI create time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.52 bool HAviFile::GetCreationDateTime (LPTSTR *datetime,* UINT * *length*)

Set AVI create time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.53 bool HAviFile::SetDateTimeOriginal (LPCTSTR *datetime*)

Set AVI orignal create date time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.54 bool HAviFile::GetDateTimeOriginal (LPTSTR *datetime,* UINT * *length*)

Get AVI orignal create date time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.55 bool HAviFile::SetAuthor (LPCTSTR *author*)

Set AVI Author infomation.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.56 bool HAviFile::GetAuthor (LPTSTR *author,* UINT * *length*)

Get AVI Author infomation.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.57 bool HAviFile::SetSoftwareName (LPCTSTR *name*)

Set software name.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.19.2.58 bool HAviFile::GetSoftwareName (LPTSTR *name*, UINT * *length*)

Get software name.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

3.20 HAviFileCallbacks Class Reference

This class handles the callbacks sent by the HAviFile playback functions.

```
#include <HAviFileCallbacks.h>
```

Public Member Functions

- virtual void OnAviImageRead (HImage *image, unsigned int index, unsigned int userData=0, void *userPtr=0)

  This function is called when an image was just read from the AVI.
- virtual void OnAviEndPlayback (unsigned int userData=0, void *userPtr=0)

  This function is called after the last image of the AVI is shown.

3.20.1 Detailed Description

This class handles the callbacks sent by the HAviFile playback functions.

Make a class inheriting from HAviFileCallbacks and override the callback functions you want to act on. Don't call any HAviFile functions from those because it could cause a deadlock.

3.20.2 Member Function Documentation

3.20.2.1 virtual void HAviFileCallbacks::OnAviImageRead (HImage * *image*, unsigned int *index*, unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called when an image was just read from the AVI.

Parameters

out	*image*	The image that was just read from the AVI file.
out	*index*	The index of the image that was just read from the AVI file.
out	*userData*	The value set in the SetCallbacks function.
out	*userPtr*	The value set in the SetCallbacks function.

3.20.2.2 virtual void HAviFileCallbacks::OnAviEndPlayback (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called after the last image of the AVI is shown.

Parameters

out	*userData*	The value set in the SetCallbacks function.

out	*userPtr*	The value set in the SetCallbacks function.

3.21 HBitmapOverlay Class Reference

This module can used to overlay a bitmap to an image.

`#include <HBitmapOverlay.h>`

Public Member Functions

- bool SetBitmapFile (LPCTSTR fileName)

  Set the bitmap file that is burned on the image.
- bool SetBitmapFile (LPCTSTR fileName, unsigned int x, unsigned int y, unsigned int width, unsigned int height)

  Set a part of a bitmap file that is burned on the image.
- LPCTSTR GetBitmapFile ()

  Get the bitmap file path.
- void SetTransparentcyColor (bool white)

  Set the transparency color. Only 2 transparency color can be defined: black or white.
- bool SetDisplayColorForBinaryBitmap (COLORREF color)

  Set the display color (for binary bitmap).
- bool IsBinaryBitmap (bool &bin)

  Check if the bitmap file is a monochrome bitmap.
- void SetOverlayPosition (int x, int y)

  Set the bitmap overlay position on the image.
- bool SuggestCenterPosition (int imageSizeX, int imageSizeY, int &overlayPositionX, int &overlayPositionY)

  Get the suggested overlay offsets to have a centered bitmap.
- bool SuggestBottomRightPosition (int imageSizeX, int imageSizeY, int &overlayPositionX, int &overlay↩PositionY)

  Get the suggested overlay offsets to have a bitmap in the lower-right corner.
- bool WriteBitmapOverlay (HImage *image)

  Write the bitmap overlay on the image (perform the burning).

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.21.1 Detailed Description

This module can used to overlay a bitmap to an image.

The pixels of the original image are replaced (burned) by the one specified in the bitmap file. The bitmap file can be of any size. If the size is larger than the image, bitmap will be clipped. Bitmap format can be of any type: 8 bit, 24 bit, monochrome or color. One can use either white or black color as a transparency : All pixels set with the transparency color in the bitmap file will not be burned to the returned image.

3.21.2 Member Function Documentation

3.21.2.1 void HBitmapOverlay::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.21.2.2 eHErrorCode HBitmapOverlay::GetLastError ( )

See also

HGrabModule::GetLastError

3.21.2.3 bool HBitmapOverlay::SetBitmapFile (LPCTSTR *fileName*)

Set the bitmap file that is burned on the image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*fileName*	The bitmap file path.

3.21.2.4 bool HBitmapOverlay::SetBitmapFile (LPCTSTR *fileName,* unsigned int *x,* unsigned int *y,* unsigned int *width,* unsigned int *height*)

Set a part of a bitmap file that is burned on the image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*fileName*	Bitmap file path.
in	*x*	X offset of the subbitmap.
in	*y*	Y offset of the subbitmap.
in	*width*	Width of the subbitmap.
in	*height*	Height of the subbitmap.

3.21.2.5 LPCTSTR HBitmapOverlay::GetBitmapFile ( )

Get the bitmap file path.

Returns

The bitmap file path.

3.21.2.6 void HBitmapOverlay::SetTransparentcyColor (bool *white*)

Set the transparency color. Only 2 transparency color can be defined: black or white.

Parameters

in	*white*	True to use white as the transparent color, false to use black.

3.21.2.7 bool HBitmapOverlay::SetDisplayColorForBinaryBitmap (COLORREF *color*)

Set the display color (for binary bitmap).

When used with a monochrome image, the custom color will actually represent the intensity of the red channel in the selected color plan, displayed in grey level.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*color*	the display color.

3.21.2.8 bool HBitmapOverlay::IsBinaryBitmap (bool & *bin*)

Check if the bitmap file is a monochrome bitmap.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*bin*	True if the bitmap is a binary bitmap, false otherwise

3.21.2.9 void HBitmapOverlay::SetOverlayPosition (int *x*, int *y*)

Set the bitmap overlay position on the image.

Default (0,0).

Returns

True if it was successful, false otherwise. Possible reason why the operation failed is because the image format is not supported. Contact Norpix to get the image format added.

Parameters

in	*x*	The bitmap overlay position in X.
in	*y*	The bitmpa overlay position in Y.

3.21.2.10 bool HBitmapOverlay::SuggestCenterPosition (int *imageSizeX*, int *imageSizeY*, int & *overlayPositionX*, int & *overlayPositionY*)

Get the suggested overlay offsets to have a centered bitmap.

The function use the current bitmap as the overlay.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*imageSizeX*	Target image size width.
in	*imageSizeY*	Target image size height.
out	*overlay↩ PositionX*	Suggested overlay position (X).
out	*overlay↩ PositionY*	Suggested overlay position (Y).

3.21.2.11 bool HBitmapOverlay::SuggestBottomRightPosition (int *imageSizeX,* int *imageSizeY,* int & *overlayPositionX,* int & *overlayPositionY*)

Get the suggested overlay offsets to have a bitmap in the lower-right corner.

The function use the current bitmap as the overlay.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*imageSizeX*	Target image size width.
in	*imageSizeY*	Target image size height.
out	*overlay↩ PositionX*	Suggested overlay position (X).
out	*overlay↩ PositionY*	Suggested overlay position (Y).

3.21.2.12 bool HBitmapOverlay::WriteBitmapOverlay (HImage * *image*)

Write the bitmap overlay on the image (perform the burning).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The image to burn.

3.22 HDaqDataSet Class Reference

This class represent a single DAQ data set.

```
#include <HDaqDataSet.h>
```

Public Member Functions

- int GetSampleCount ()

  Get the number of samples in the dataset.
- int GetChannelNum ()

  Get the number of channels in the dataset.

- float * GetSample (int sampleIndex)

  Get the value of a sample at a specific index.
- long GetTimeStamp ()

  Retreive the absolute time stamp of the dataset in time_t format (32-bit).
- unsigned short GetTimeStampMS ()

  Retrieve the milliseconds part of the dataset absolute time stamp.
- unsigned short GetTimeStampUS ()

  Retreive the microseconds part of the absolute dataset time stamp.
- void SetTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS=0)

  Change the time stamp of a HImage.
- void SetDaqDataSet (int channelCount, int sampleCount, float *values, long time, unsigned short timeMS, unsigned short timeUS)

  Set the content of a dataset.
- void SetDummyDataSet (bool dummy)

  Set the dataset as a dummy dataset.
- bool IsDummyDataSet ()

  Check if the dataset contains dummy values.
- int GetSampleCount ()

  Get the number of samples in the dataset.
- int GetChannelNum ()

  Get the number of channels in the dataset.
- float * GetSample (int sampleIndex)

  Get the value of a sample at a specific index.
- long GetTimeStamp ()

  Retreive the absolute time stamp of the dataset in time_t format (32-bit).
- unsigned short GetTimeStampMS ()

  Retrieve the milliseconds part of the dataset absolute time stamp.
- unsigned short GetTimeStampUS ()

  Retreive the microseconds part of the absolute dataset time stamp.
- void SetTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS=0)

  Change the time stamp of a HImage.
- void SetDaqDataSet (int channelCount, int sampleCount, float *values, long time, unsigned short timeMS, unsigned short timeUS)

  Set the content of a dataset.
- void SetDummyDataSet (bool dummy)

  Set the dataset as a dummy dataset.
- bool IsDummyDataSet ()

  Check if the dataset contains dummy values.

- void SetReferenceFrameIndex (long frameIndex)

  MEDATA FUNCTION
  Set associated reference video frame index.
- int GetReferenceFrameIndex ()

  Get associated reference video frame index.

- void SetReferenceFrameIndex (long frameIndex)

  MEDATA FUNCTION
  Set associated reference video frame index.
- int GetReferenceFrameIndex ()

  Get associated reference video frame index.

Static Public Member Functions

- static HDaqDataSet ∗ CreateDaqDataSet ()

  Create an empty HDaqDataSet in which a dataset from a daq file can be read.
- static HDaqDataSet ∗ CloneDaqDataSet (HDaqDataSet ∗srcDataSet)

  Create a perfect copy of a source HDaqDataSet.
- static void CloneDaqDataSet (HDaqDataSet ∗srcDataSet, HDaqDataSet ∗destDataSet)

  Create a perfect copy of a dataset.
- static void DeleteDaqDataSet (HDaqDataSet ∗dataSet)

  Delete an HDaqDataSet that was allocated with a call to CreateDaqDataSet().
- static HDaqDataSet ∗ CreateDaqDataSet ()

  Create an empty HDaqDataSet in which a dataset from a daq file can be read.
- static HDaqDataSet ∗ CloneDaqDataSet (HDaqDataSet ∗srcDataSet)

  Create a perfect copy of a source HDaqDataSet.
- static void CloneDaqDataSet (HDaqDataSet ∗srcDataSet, HDaqDataSet ∗destDataSet)

  Create a perfect copy of a dataset.
- static void DeleteDaqDataSet (HDaqDataSet ∗dataSet)

  Delete an HDaqDataSet that was allocated with a call to CreateDaqDataSet().

3.22.1 Detailed Description

This class represent a single DAQ data set.

3.22.2 Member Function Documentation

3.22.2.1 static HDaqDataSet∗ HDaqDataSet::CreateDaqDataSet ( ) [static]

Create an empty HDaqDataSet in which a dataset from a daq file can be read.

When it is no longer needed, free it by using DeleteDaqDataSet.

Returns

A pointer to a newly created HDaqDataSet.

3.22.2.2 static HDaqDataSet∗ HDaqDataSet::CloneDaqDataSet (HDaqDataSet ∗ *srcDataSet*) [static]

Create a perfect copy of a source HDaqDataSet.

When it is no longer needed, free it by using DeleteDaqDataSet.

Returns

A pointer to the newly created HDaqDataSet.

Parameters

in	*srcDataSet*	The source dataset to clone.

3.22.2.3 static void HDaqDataSet::CloneDaqDataSet (HDaqDataSet ∗ *srcDataSet,* HDaqDataSet ∗ *destDataSet*) [static]

Create a perfect copy of a dataset.

For the destDataSet, only use an HDaqDataSet you created using HDaqDataSet::CreateDaqDataSet. If there is currently any data in the destDataSet, it will be deleted prior to the copy.

Parameters

in	*srcDataSet*	The source dataset to clone.
in	*destDataSet*	The destination dataset.

3.22.2.4 static void HDaqDataSet::DeleteDaqDataSet (HDaqDataSet * *dataSet*) [static]

Delete an HDaqDataSet that was allocated with a call to CreateDaqDataSet().

Only user-created HDaqDataSet should be deleted.

Parameters

in	*dataSet*	The HDaqDataSet to delete.

3.22.2.5 void HDaqDataSet::SetReferenceFrameIndex (long *frameIndex*)

MEDATA FUNCTION

Set associated reference video frame index.

3.22.2.6 int HDaqDataSet::GetSampleCount ( )

Get the number of samples in the dataset.

Returns

The sample count.

3.22.2.7 int HDaqDataSet::GetChannelNum ( )

Get the number of channels in the dataset.

Each channel has (GetSampleCount() / GetChannelNum()) samples in the dataset.

Returns

The channel count.

3.22.2.8 float* HDaqDataSet::GetSample (int *sampleIndex*)

Get the value of a sample at a specific index.

Returns

the sample value.

Parameters

in	*sampleIndex*	The sample index

3.22.2.9 long HDaqDataSet::GetTimeStamp ( )

Retreive the absolute time stamp of the dataset in time_t format (32-bit).

More precision can be obtained by calling GetTimestampMS/US.

Returns

The absolute time stamp in seconds.

3.22.2.10 unsigned short HDaqDataSet::GetTimeStampMS ( )

Retrieve the milliseconds part of the dataset absolute time stamp.

Depending on the latency of the operating system, the precision might be off by 1 to 3 milliseconds.

Returns

The milliseconds part of the absolute dataset timestamp.

3.22.2.11 unsigned short HDaqDataSet::GetTimeStampUS ( )

Retreive the microseconds part of the absolute dataset time stamp.

This value will be 0 unless the dataset was time stamped with an external high performance timer device used in conjunction with the data acquisition device.

Returns

The microsecond part of the dataset time stamp.

3.22.2.12 void HDaqDataSet::SetTimeStamp (long *timeStamp*, unsigned short *timeStampMS*, unsigned short *timeStampUS* = 0)

Change the time stamp of a HImage.

Parameters

in	*timeStamp*	The absolute time stamp in seconds (time_t).
in	*timeStampMS*	The milliseconds part of the image time stamp.
in	*timeStampUS*	The microseconds part of the image time stamp.

3.22.2.13 void HDaqDataSet::SetDaqDataSet (int *channelCount*, int *sampleCount*, float * *values*, long *time*, unsigned short *timeMS*, unsigned short *timeUS*)

Set the content of a dataset.

Parameters

in	*channelCount*	The channel count.
in	*sampleCount*	The sample count.
in	*values*	An array of samples.
in	*time*	The absolute time stamp in seconds (time_t).
in	*timeMS*	The milliseconds part of the image time stamp.
in	*timeUS*	The microseconds part of the image time stamp.

3.22.2.14 void HDaqDataSet::SetDummyDataSet (bool *dummy*)

Set the dataset as a dummy dataset.

Parameters

in	*dummy*	True if the data set contains dummy values

3.22.2.15 bool HDaqDataSet::IsDummyDataSet ( )

Check if the dataset contains dummy values.

Returns

True if it is a dummy data set, false otherwise.

3.22.2.16 static HDaqDataSet* HDaqDataSet::CreateDaqDataSet ( ) [static]

Create an empty HDaqDataSet in which a dataset from a daq file can be read.

When it is no longer needed, free it by using DeleteDaqDataSet.

Returns

A pointer to a newly created HDaqDataSet.

3.22.2.17 static HDaqDataSet* HDaqDataSet::CloneDaqDataSet (HDaqDataSet * *srcDataSet*) [static]

Create a perfect copy of a source HDaqDataSet.

When it is no longer needed, free it by using DeleteDaqDataSet.

Returns

A pointer to the newly created HDaqDataSet.

Parameters

in	*srcDataSet*	The source dataset to clone.

3.22.2.18 static void HDaqDataSet::CloneDaqDataSet (HDaqDataSet * *srcDataSet,* HDaqDataSet * *destDataSet*) [static]

Create a perfect copy of a dataset.

For the destDataSet, only use an HDaqDataSet you created using HDaqDataSet::CreateDaqDataSet. If there is currently any data in the destDataSet, it will be deleted prior to the copy.

Parameters

in	*srcDataSet*	The source dataset to clone.
in	*destDataSet*	The destination dataset.

3.22.2.19 static void HDaqDataSet::DeleteDaqDataSet (HDaqDataSet * *dataSet*) [static]

Delete an HDaqDataSet that was allocated with a call to CreateDaqDataSet().

Only user-created HDaqDataSet should be deleted.

Parameters

in	*dataSet*	The HDaqDataSet to delete.

3.22.2.20 void HDaqDataSet::SetReferenceFrameIndex (long *frameIndex*)

MEDATA FUNCTION

Set associated reference video frame index.

3.22.2.21 int HDaqDataSet::GetSampleCount ( )

Get the number of samples in the dataset.

Returns

The sample count.

3.22.2.22 int HDaqDataSet::GetChannelNum ( )

Get the number of channels in the dataset.

Each channel has (GetSampleCount() / GetChannelNum()) samples in the dataset.

Returns

The channel count.

3.22.2.23 float* HDaqDataSet::GetSample (int *sampleIndex*)

Get the value of a sample at a specific index.

Returns

the sample value.

Parameters

in	*sampleIndex*	The sample index

3.22.2.24 long HDaqDataSet::GetTimeStamp ( )

Retreive the absolute time stamp of the dataset in time_t format (32-bit).

More precision can be obtained by calling GetTimestampMS/US.

Returns

The absolute time stamp in seconds.

3.22.2.25 unsigned short HDaqDataSet::GetTimeStampMS ( )

Retrieve the milliseconds part of the dataset absolute time stamp.

Depending on the latency of the operating system, the precision might be off by 1 to 3 milliseconds.

Returns

The milliseconds part of the absolute dataset timestamp.

3.22.2.26 unsigned short HDaqDataSet::GetTimeStampUS ( )

Retreive the microseconds part of the absolute dataset time stamp.

This value will be 0 unless the dataset was time stamped with an external high performance timer device used in conjunction with the data acquisition device.

Returns

The microsecond part of the dataset time stamp.

3.22.2.27 void HDaqDataSet::SetTimeStamp (long *timeStamp*, unsigned short *timeStampMS*, unsigned short *timeStampUS* = 0)

Change the time stamp of a HImage.

Parameters

in	*timeStamp*	The absolute time stamp in seconds (time_t).
in	*timeStampMS*	The milliseconds part of the image time stamp.
in	*timeStampUS*	The microseconds part of the image time stamp.

3.22.2.28 void HDaqDataSet::SetDaqDataSet (int *channelCount*, int *sampleCount*, float * *values*, long *time*, unsigned short *timeMS*, unsigned short *timeUS*)

Set the content of a dataset.

Parameters

in	*channelCount*	The channel count.
in	*sampleCount*	The sample count.
in	*values*	An array of samples.
in	*time*	The absolute time stamp in seconds (time_t).
in	*timeMS*	The milliseconds part of the image time stamp.
in	*timeUS*	The microseconds part of the image time stamp.

3.22.2.29 void HDaqDataSet::SetDummyDataSet (bool *dummy*)

Set the dataset as a dummy dataset.

Parameters

in	*dummy*	True if the data set contains dummy values

3.22.2.30 bool HDaqDataSet::IsDummyDataSet ( )

Check if the dataset contains dummy values.

Returns

True if it is a dummy data set, false otherwise.

3.23 HDaqFile Class Reference

This class handle a DAQ file and all related operations.

```
#include <HDaqFile.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()
- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

DAQ FILE MANAGEMENT FUNCTIONS

- bool Create (LPCTSTR daqPath)
  Create a new DAQ file.
- bool Open (LPCTSTR daqPath, bool &readOnly)
  Open an existing DAQ file.
- bool Close ()
  Close the current DAQ file.
- bool Create (LPCTSTR daqPath)
  Create a new DAQ file.
- bool Open (LPCTSTR daqPath, bool &readOnly)
  Open an existing DAQ file.
- bool Close ()
  Close the current DAQ file.

DAQ FILE READING/WRITING FUNCTIONS

- bool Write (HDaqDataSet *dataset)
  Write the specified dataset in the DAQ file at the current recording position.
- bool Read (HDaqDataSet *dataSet, unsigned __int64 index)
  Read a dataset in the current DAQ file (on disk or in RAM).
- bool Read (HDaqDataSet *dataSet)
  Read the dataset at the current reading position from the DAQ file.
- bool Read (HDaqDataSet *dataSet, long timeStamp, unsigned short timeStampMS, unsigned short time↩StampUS)
  Read the dataset which is the closest to a given time from the DAQ file.
- unsigned __int64 GetAllocatedDataSets ()
  Retrieve the number of dataset stored in the current DAQ file.

- unsigned __int64 GetRecordingPosition ()

  Retrieve the position where the next dataset will be written in the DAQ file.
- unsigned __int64 GetReadingPosition ()

  Retrieve the position where the next dataset will be read from.
- int GetSampleRate ()

  Retrieve sample rate.
- void SetSampleRate (int rate)

  Set sample rate.
- unsigned __int64 GetRecordingPositionByRefFrame (unsigned long frameIndex)

  Get position by reference video frameIndex.
- unsigned __int64 GetRecordingPositionByRefTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS)

  Retrieve position by time stamp.
- bool GetReferenceInfo (unsigned __int64 index, unsigned long &refFrameIndex, long &timeStamp, unsigned short &timeStampMS, unsigned short &timeStampUS)

  Retrieve video reference infomation from specified daq data set.
- long GetSamplesPerDataSet ()

  Retrieve number of samplesper daq data set.
- long GetDaqChannelCount ()

  Retrieve number of channels.
- bool SetReadingPosition (unsigned __int64 postion)

  Set read position.
- unsigned __int64 GetPositionAtTime (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS)

  Search the daq file to find the last frame that had been captured at a given time.
- bool GetTimeAtPosition (unsigned __int64 index, long &timeStamp, unsigned short &timeStampMS, unsigned short &timeStampUS)

  Retrieve the timestamp of a given frame in the daq file.
- bool Write (HDaqDataSet *dataset)

  Write the specified dataset in the DAQ file at the current recording position.
- bool Read (HDaqDataSet *dataSet, unsigned __int64 index)

  Read a dataset in the current DAQ file (on disk or in RAM).
- bool Read (HDaqDataSet *dataSet)

  Read the dataset at the current reading position from the DAQ file.
- bool Read (HDaqDataSet *dataSet, long timeStamp, unsigned short timeStampMS, unsigned short time↩StampUS)

  Read the dataset which is the closest to a given time from the DAQ file.
- unsigned __int64 GetAllocatedDataSets ()

  Retrieve the number of dataset stored in the current DAQ file.
- unsigned __int64 GetRecordingPosition ()

  Retrieve the position where the next dataset will be written in the DAQ file.
- unsigned __int64 GetReadingPosition ()

  Retrieve the position where the next dataset will be read from.
- int GetSampleRate ()

  Retrieve sample rate.
- void SetSampleRate (int rate)

  Set sample rate.
- unsigned __int64 GetRecordingPositionByRefFrame (unsigned long frameIndex)

  Get position by reference video frameIndex.
- unsigned __int64 GetRecordingPositionByRefTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS)

  Retrieve position by time stamp.
- bool GetReferenceInfo (unsigned __int64 index, unsigned long &refFrameIndex, long &timeStamp, unsigned short &timeStampMS, unsigned short &timeStampUS)

  Retrieve video reference infomation from specified daq data set.
- long GetSamplesPerDataSet ()

  Retrieve number of samplesper daq data set.
- long GetDaqChannelCount ()

Retrieve number of channels.
- bool SetReadingPosition (unsigned __int64 postion)
  Set read position.
- unsigned __int64 GetPositionAtTime (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS)
  Search the daq file to find the last frame that had been captured at a given time.
- bool GetTimeAtPosition (unsigned __int64 index, long &timeStamp, unsigned short &timeStampMS, unsigned short &timeStampUS)
  Retrieve the timestamp of a given frame in the daq file.

PLAYBACK FUNCTIONS

- void SetPlaybackPosition (unsigned __int64 index)
  Set the start position of the playback.
- unsigned __int64 GetPlaybackPosition ()
  Get the current playback position.
- bool StartPlayback ()
  Initiates playback of a DAQ file from its current playback position.
- bool StopPlayback ()
  Stop DAQ file playback.
- bool IsPlaying ()
  Retrieve the playback state, i.e.
- void ResynchronizePlayback (long offset, short offsetMS)
  Put an offset on the start time reference of a timestamped DAQ file playback.
- void SetPlaybackPosition (unsigned __int64 index)
  Set the start position of the playback.
- unsigned __int64 GetPlaybackPosition ()
  Get the current playback position.
- bool StartPlayback ()
  Initiates playback of a DAQ file from its current playback position.
- bool StopPlayback ()
  Stop DAQ file playback.
- bool IsPlaying ()
  Retrieve the playback state, i.e.
- void ResynchronizePlayback (long offset, short offsetMS)
  Put an offset on the start time reference of a timestamped DAQ file playback.

CALLBACKS RELATED FUNCTIONS

- bool **SetCallbacks** (HDaqFileCallbacks *callbacks)
- bool **SetCallbacks** (HDaqFileCallbacks *callbacks)

META DATA MANAGEMENT FUNCTIONS

- bool SetMetadataSize (int metadataSizeBytes)
  Set total meda data bufffer size.
- bool RemoveMetadata (long id)
  Remove meda data by ID.
- bool SetMetadata (long id, int length, void *data)
  Add meda data.
- int GetMetadataLength (long id)
  Get meda data size.
- bool GetMetadata (long id, void *data)
  Read meda data.
- bool SetMetadataSize (int metadataSizeBytes)
  Set total meda data bufffer size.
- bool RemoveMetadata (long id)
  Remove meda data by ID.
- bool SetMetadata (long id, int length, void *data)
  Add meda data.
- int GetMetadataLength (long id)
  Get meda data size.
- bool GetMetadata (long id, void *data)
  Read meda data.

3.23.1 Detailed Description

This class handle a DAQ file and all related operations.

3.23.2 Member Function Documentation

3.23.2.1 void HDaqFile::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.23.2.2 eHErrorCode HDaqFile::GetLastError ( )

See also

HGrabModule::GetLastError

3.23.2.3 bool HDaqFile::Create (LPCTSTR *daqPath*)

Create a new DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*daqPath*	The full path where the DAQ file is to be saved (ex: "c:\capture.daq"). Any file with the same name will be overwritten. If daqPath is empty, the file is created in RAM.

3.23.2.4 bool HDaqFile::Open (LPCTSTR *daqPath,* bool & *readOnly*)

Open an existing DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*daqPath*	The full path where the file is located. (ex: "c:\capture.daq").
in,out	*readOnly*	True to open the file in read-only, false otherwise.

3.23.2.5 bool HDaqFile::Close ( )

Close the current DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.23.2.6 bool HDaqFile::Write (HDaqDataSet * *dataset*)

Write the specified dataset in the DAQ file at the current recording position.

SetRecordingPosition() can be used to set a specific recording position, overwriting an older dataset. Once the function completes, the recording position will be set to append the next dataset after the current one.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

dataset	The dataset to be saved to the DAQ file.

3.23.2.7 bool HDaqFile::Read (HDaqDataSet * *dataSet*, unsigned __int64 *index*)

Read a dataset in the current DAQ file (on disk or in RAM).

A HDaqDataSet will need to be allocated to be read and freed once it is done with. Refer to the example below.

Example :

```
HDaqFile file;
//Creates or opens a DAQ file.  The file must not be empty.
...

//Allocate a HDaqDataSet to hold the data that will be read.
HDaqDataSet* dataset = HDaqDataSet::CreateDaqDataSet();

//Read the first dataset from the DAQ file
file.Read(dataset, 1);

//Display the dataset or processes it
...

//Deallocate the dataset
HDaqDataSet::DeleteDaqDataSet(dataset);
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*dataSet*	A dataset to hold the data from the DAQ file.
in	*index*	The requested dataset index. This ranges from 1 to the value retrieved with GetAllocatedDataSets().

3.23.2.8 bool HDaqFile::Read (HDaqDataSet * *dataSet*)

Read the dataset at the current reading position from the DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*dataSet*	A dataset to hold the data from the DAQ file.

3.23.2.9 bool HDaqFile::Read (HDaqDataSet * *dataSet,* long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Read the dataset which is the closest to a given time from the DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*dataSet*	A dataset to hold the data from the DAQ file.
in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.10 unsigned __int64 HDaqFile::GetAllocatedDataSets ( )

Retrieve the number of dataset stored in the current DAQ file.

Returns

The number of dataset in the DAQ file.

3.23.2.11 unsigned __int64 HDaqFile::GetRecordingPosition ( )

Retrieve the position where the next dataset will be written in the DAQ file.

Returns

The current recording position.

3.23.2.12 unsigned __int64 HDaqFile::GetReadingPosition ( )

Retrieve the position where the next dataset will be read from.

Returns

The current playback position.

3.23.2.13 int HDaqFile::GetSampleRate ( )

Retrieve sample rate.

Returns

sample rate.

3.23.2.14 unsigned __int64 HDaqFile::GetRecordingPositionByRefFrame (unsigned long *frameIndex*)

Get position by reference video frameIndex.

Returns

Non-zero if DAQ data position otherwise no reference video frameIndex find.

Parameters

in	*frameIndex*	reference video frame index.

3.23.2.15 unsigned __int64 HDaqFile::GetRecordingPositionByRefTimeStamp (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Retrieve position by time stamp.

Returns

Non-zero if DAQ data position otherwise no daq data find.

Parameters

in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.16 bool HDaqFile::GetReferenceInfo (unsigned __int64 *index,* unsigned long & *refFrameIndex,* long & *timeStamp,* unsigned short & *timeStampMS,* unsigned short & *timeStampUS*)

Retrieve video reference infomation from specified daq data set.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	Daq data index to query.
in	*refFrameIndex*	reference video frame index.
in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.17 unsigned __int64 HDaqFile::GetPositionAtTime (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Search the daq file to find the last frame that had been captured at a given time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.18 bool HDaqFile::GetTimeAtPosition (unsigned __int64 *index*, long & *timeStamp*, unsigned short & *timeStampMS*, unsigned short & *timeStampUS*)

Retrieve the timestamp of a given frame in the daq file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The dataset index.
in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.19 void HDaqFile::SetPlaybackPosition (unsigned __int64 *index*)

Set the start position of the playback.

See also

StartPlayback()

Parameters

index	The index of the starting dataset.

3.23.2.20 unsigned __int64 HDaqFile::GetPlaybackPosition ( )

Get the current playback position.

Returns

The index of the dataset at the current playback position.

3.23.2.21 bool HDaqFile::StartPlayback ( )

Initiates playback of a DAQ file from its current playback position.

The current position can be retrieved and set with the GetPlaybackPosition and SetPlaybackPosition functions before calling Play.

The playback is done according to the timestamp of each dataset. Datasets will be sent to the callbacks functions.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Example :

```
//Starting a playback from the beginning of the current DAQ file.
file.SetPlaybackPosition(1);
file.StartPlayback();
```

3.23.2.22 bool HDaqFile::StopPlayback ( )

Stop DAQ file playback.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.23.2.23 bool HDaqFile::IsPlaying ( )

Retrieve the playback state, i.e.

playing or not. Example :

```
HDaqFile file;

void TogglePlayback()
{
   if(file.IsPlaying())
         file.Stop();
   else
         file.Play();
}
```

Returns

True if playback is running, false if playback stopped.

3.23.2.24 void HDaqFile::ResynchronizePlayback (long *offset*, short *offsetMS*)

Put an offset on the start time reference of a timestamped DAQ file playback.

Sending a negative offset will make the playback accellerate. For example, a "-3 seconds" offset tells the playback that it is 3 second late and must do some catch-up. A positive offset will make the playback stalls for the specified amount of time.

Parameters

in	*offset*	The offset to apply (in seconds).
in	*offsetMS*	The offset to apply (milliseconds part)

3.23.2.25 bool HDaqFile::SetMetadataSize (int *metadataSizeBytes*)

Set total meda data bufffer size.

Parameters

in	*metadataSize↩ Bytes*	buffer size.

3.23.2.26 bool HDaqFile::RemoveMetadata (long *id*)

Remove meda data by ID.

Parameters

in	*id*	meda data ID.

3.23.2.27 bool HDaqFile::SetMetadata (long *id*, int *length*, void * *data*)

Add meda data.

Parameters

in	*id*	meda data ID.
in	*length*	meda data buffer size.
in	*data*	meda data buffer.

3.23.2.28 int HDaqFile::GetMetadataLength (long *id*)

Get meda data size.

Parameters

id	...

3.23.2.29 bool HDaqFile::GetMetadata (long *id*, void * *data*)

Read meda data.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*id*	meda data ID.
out	*data*	meda data.

3.23.2.30 void HDaqFile::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.23.2.31 eHErrorCode HDaqFile::GetLastError ( )

See also

HGrabModule::GetLastError

3.23.2.32 bool HDaqFile::Create (LPCTSTR *daqPath*)

Create a new DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*daqPath*	The full path where the DAQ file is to be saved (ex: "c:\capture.daq"). Any file with the same name will be overwritten. If daqPath is empty, the file is created in RAM.

3.23.2.33 bool HDaqFile::Open (LPCTSTR *daqPath,* bool & *readOnly*)

Open an existing DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*daqPath*	The full path where the file is located. (ex: "c:\capture.daq").
in,out	*readOnly*	True to open the file in read-only, false otherwise.

3.23.2.34 bool HDaqFile::Close ( )

Close the current DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.23.2.35 bool HDaqFile::Write (HDaqDataSet * *dataset*)

Write the specified dataset in the DAQ file at the current recording position.

SetRecordingPosition() can be used to set a specific recording position, overwriting an older dataset. Once the function completes, the recording position will be set to append the next dataset after the current one.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

dataset	The dataset to be saved to the DAQ file.

3.23.2.36 bool HDaqFile::Read (HDaqDataSet * *dataSet,* unsigned __int64 *index*)

Read a dataset in the current DAQ file (on disk or in RAM).

A HDaqDataSet will need to be allocated to be read and freed once it is done with. Refer to the example below.

Example :

```
HDaqFile file;
//Creates or opens a DAQ file.  The file must not be empty.
...

//Allocate a HDaqDataSet to hold the data that will be read.
```

```
HDaqDataSet* dataset = HDaqDataSet::CreateDaqDataSet();

//Read the first dataset from the DAQ file
file.Read(dataset, 1);

//Display the dataset or processes it
...

//Deallocate the dataset
HDaqDataSet::DeleteDaqDataSet(dataset);
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*dataSet*	A dataset to hold the data from the DAQ file.
in	*index*	The requested dataset index. This ranges from 1 to the value retrieved with GetAllocatedDataSets().

3.23.2.37 bool HDaqFile::Read (HDaqDataSet * *dataSet*)

Read the dataset at the current reading position from the DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*dataSet*	A dataset to hold the data from the DAQ file.

3.23.2.38 bool HDaqFile::Read (HDaqDataSet * *dataSet*, long *timeStamp*, unsigned short *timeStampMS*, unsigned short *timeStampUS*)

Read the dataset which is the closest to a given time from the DAQ file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*dataSet*	A dataset to hold the data from the DAQ file.
in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.39 unsigned __int64 HDaqFile::GetAllocatedDataSets ( )

Retrieve the number of dataset stored in the current DAQ file.

Returns

The number of dataset in the DAQ file.

3.23.2.40 unsigned __int64 HDaqFile::GetRecordingPosition ( )

Retrieve the position where the next dataset will be written in the DAQ file.

Returns

The current recording position.

3.23.2.41 unsigned __int64 HDaqFile::GetReadingPosition ( )

Retrieve the position where the next dataset will be read from.

Returns

The current playback position.

3.23.2.42 int HDaqFile::GetSampleRate ( )

Retrieve sample rate.

Returns

sample rate.

3.23.2.43 unsigned __int64 HDaqFile::GetRecordingPositionByRefFrame (unsigned long *frameIndex*)

Get position by reference video frameIndex.

Returns

Non-zero if DAQ data position otherwise no reference video frameIndex find.

Parameters

in	*frameIndex*	reference video frame index.

3.23.2.44 unsigned __int64 HDaqFile::GetRecordingPositionByRefTimeStamp (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Retrieve position by time stamp.

Returns

Non-zero if DAQ data position otherwise no daq data find.

Parameters

in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.

in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.45 bool HDaqFile::GetReferenceInfo (unsigned __int64 *index,* unsigned long & *refFrameIndex,* long & *timeStamp,* unsigned short & *timeStampMS,* unsigned short & *timeStampUS*)

Retrieve video reference infomation from specified daq data set.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	Daq data index to query.
in	*refFrameIndex*	reference video frame index.
in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.46 unsigned __int64 HDaqFile::GetPositionAtTime (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Search the daq file to find the last frame that had been captured at a given time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.47 bool HDaqFile::GetTimeAtPosition (unsigned __int64 *index,* long & *timeStamp,* unsigned short & *timeStampMS,* unsigned short & *timeStampUS*)

Retrieve the timestamp of a given frame in the daq file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The dataset index.
in	*timeStamp*	The absolute timestamp in seconds (32-bit time_t).

in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.23.2.48 void HDaqFile::SetPlaybackPosition (unsigned __int64 *index*)

Set the start position of the playback.

See also

StartPlayback()

Parameters

index	The index of the starting dataset.

3.23.2.49 unsigned __int64 HDaqFile::GetPlaybackPosition ( )

Get the current playback position.

Returns

The index of the dataset at the current playback position.

3.23.2.50 bool HDaqFile::StartPlayback ( )

Initiates playback of a DAQ file from its current playback position.

The current position can be retrieved and set with the GetPlaybackPosition and SetPlaybackPosition functions before calling Play.

The playback is done according to the timestamp of each dataset. Datasets will be sent to the callbacks functions.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Example :

```
//Starting a playback from the beginning of the current DAQ file.
file.SetPlaybackPosition(1);
file.StartPlayback();
```

3.23.2.51 bool HDaqFile::StopPlayback ( )

Stop DAQ file playback.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.23.2.52 bool HDaqFile::IsPlaying ( )

Retrieve the playback state, i.e.

playing or not. Example :

```
HDaqFile file;

void TogglePlayback()
{
   if(file.IsPlaying())
         file.Stop();
   else
         file.Play();
}
```

Returns

True if playback is running, false if playback stopped.

3.23.2.53 void HDaqFile::ResynchronizePlayback (long *offset*, short *offsetMS*)

Put an offset on the start time reference of a timestamped DAQ file playback.

Sending a negative offset will make the playback accellerate. For example, a "-3 seconds" offset tells the playback that it is 3 second late and must do some catch-up. A positive offset will make the playback stalls for the specified amount of time.

Parameters

in	*offset*	The offset to apply (in seconds).
in	*offsetMS*	The offset to apply (milliseconds part)

3.23.2.54 bool HDaqFile::SetMetadataSize (int *metadataSizeBytes*)

Set total meda data bufffer size.

Parameters

in	*metadataSize↩ Bytes*	buffer size.

3.23.2.55 bool HDaqFile::RemoveMetadata (long *id*)

Remove meda data by ID.

Parameters

in	*id*	meda data ID.

3.23.2.56 bool HDaqFile::SetMetadata (long *id*, int *length*, void * *data*)

Add meda data.

Parameters

in	*id*	meda data ID.
in	*length*	meda data buffer size.
in	*data*	meda data buffer.

3.23.2.57 int HDaqFile::GetMetadataLength (long *id*)

Get meda data size.

Parameters

id	...

3.23.2.58 bool HDaqFile::GetMetadata (long *id*, void * *data*)

Read meda data.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*id*	meda data ID.
out	*data*	meda data.

3.24 HDaqFileCallbacks Class Reference

This class manages the DAQ playback's callbacks.

```
#include <HDaqCallbacks.h>
```

Public Member Functions

- virtual void OnDaqPlaybackUpdate (unsigned __int64 updateIndex)

  This function is called when a dataset is "played" from a DAQ file during playback.
- virtual void OnEndOfDaqFile ()

  This function is called when the last DAQ dataset of the file is reached.
- virtual void OnDaqPlaybackStopped ()

  This function is called when the sequence playback stops.
- virtual void OnDaqPlaybackUpdate (unsigned __int64 updateIndex)

  This function is called when a dataset is "played" from a DAQ file during playback.
- virtual void OnEndOfDaqFile ()

  This function is called when the last DAQ dataset of the file is reached.
- virtual void OnDaqPlaybackStopped ()

  This function is called when the sequence playback stops.

3.24.1 Detailed Description

This class manages the DAQ playback's callbacks.

Make a class inheriting from HSequencerCallbacks and override the callback functions you want to act on.

3.24.2 Member Function Documentation

3.24.2.1 virtual void HDaqFileCallbacks::OnDaqPlaybackUpdate (unsigned __int64 *updateIndex*) [inline], [virtual]

This function is called when a dataset is "played" from a DAQ file during playback.

Parameters

out	*updateIndex*	The current dataset index.

3.24.2.2 virtual void HDaqFileCallbacks::OnDaqPlaybackUpdate (unsigned __int64 *updateIndex*) [inline], [virtual]

This function is called when a dataset is "played" from a DAQ file during playback.

Parameters

out	*updateIndex*	The current dataset index.

3.25 HDeliverCallback Class Reference

Allows the user to be notified before a image is going to be compressed and add to an H.264 stream.

`#include <HStreamCallback.h>`

Public Member Functions

- virtual HImage * OnPrepareCompress (HImage *src, HImage *target)

  This function is called before a image is going to be compressed.

3.25.1 Detailed Description

Allows the user to be notified before a image is going to be compressed and add to an H.264 stream.

User can use this function to do color pre-processing. By default this function return source image. Make a class inheriting from HDeliverCallback and override the callback functions you want to act on.

3.25.2 Member Function Documentation

3.25.2.1 virtual HImage* HDeliverCallback::OnPrepareCompress (HImage * *src*, HImage * *target*) [inline], [virtual]

This function is called before a image is going to be compressed.

Returns

The image to be compressed

Parameters

in	*src*	source image.
out	*target*	processed image buffer.

3.26 HErrorManager Class Reference

This class translate Hermes error codes to text strings.

`#include <HErrorManager.h>`

Static Public Member Functions

- static LPCTSTR GetErrorString (eHErrorCode code)

  Get the string representation of an error code.

3.26.1 Detailed Description

This class translate Hermes error codes to text strings.

3.26.2 Member Function Documentation

3.26.2.1 static LPCTSTR HErrorManager::GetErrorString (eHErrorCode *code*) [inline],[static]

Get the string representation of an error code.

Returns

The string of the error.

Parameters

in	*code*	The error code.

3.27 HFPSMonitor Class Reference

This class can be used to easily compute a frame rate.

`#include <HFPSMonitor.h>`

Public Member Functions

- bool SetCallbacks (HFPSMonitorCallbacks *userCallbacks, unsigned int userData=0, void *userPtr=NULL)

  SetCallbacks sets the callback class used by the monitor.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

FPS FUNCTIONS

- bool StartLiveCheck (unsigned int updateInterval=1000)

  Updates the FPS at the desired milliseconds interval.
- bool NewLiveFrame (bool forceUpdate)

  Tell the monitor that a new frame was just received.

MANUAL BENCH FUNCTION

- bool StartBench ()

  Start the benchmark.
- bool EndBench (unsigned long frameCount, double &fps)

  Stop the benchmark.
- bool GetBenchTime (long &seconds, unsigned short &ms)

  Get the bench times.

3.27.1 Detailed Description

This class can be used to easily compute a frame rate.

It can be used to monitor the frames per second speed of anything, be it capture, playback or even non image related content.

Typical usage :

```
HFPSMonitor FPSMonitor;

//Set the function to be called at each fps update.
FPSMonitor.SetCallbacks(...);

//Set the FPS monitor to update us with the new fps result every 1000 ms
FPSMonitor.StartLiveCheck(1000);

Every time a new frame is received from the grabber, tell the monitor
FPSMonitor.NewLiveFrame(false); //new frame received
```

3.27.2 Member Function Documentation

3.27.2.1 void HFPSMonitor::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.27.2.2 eHErrorCode HFPSMonitor::GetLastError ( )

See also

HGrabModule::GetLastError

3.27.2.3 bool HFPSMonitor::StartLiveCheck (unsigned int *updateInterval* = 1000)

Updates the FPS at the desired milliseconds interval.

Every updateInterval, the class will launch the OnNewFrame callback.

See also

SetCallbacks

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*updateInterval*	The update interval in milliseconds.

3.27.2.4 bool HFPSMonitor::NewLiveFrame (bool *forceUpdate*)

Tell the monitor that a new frame was just received.

The monitor will then recompute the FPS value and launch the OnNewFrame callback if the updateInterval time is reached.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*forceUpdate*	If true, the OnNewFrame callback will be launch regardless of the update↩ Interval set.

3.27.2.5 bool HFPSMonitor::StartBench ( )

Start the benchmark.

Used in conjunction with EndBench, StartBench is used to compute the frame rate over a given time period. Call StartBench to save the current time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.27.2.6 bool HFPSMonitor::EndBench (unsigned long *frameCount*, double & *fps*)

Stop the benchmark.

Compute the frame rate given a specific number of frames. StartBench must have been called at least once prior to calling EndBench.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*frameCount*	The number of frames captured since StartBench has been called.
out	*fps*	The frame rate. fps = frameCount / ((EndBench Time) - (StartBench Time)).

3.27.2.7 bool HFPSMonitor::GetBenchTime (long & *seconds*, unsigned short & *ms*)

Get the bench times.

Compute the time elapsed since the bench was started. StartBench must have been called at least once prior to calling GetBenchTime.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*seconds*	The number of seconds elapsed since StartBench has been called.

out	*ms*	The milliseconds part of the time elapsed.

3.27.2.8 bool HFPSMonitor::SetCallbacks (HFPSMonitorCallbacks * *userCallbacks*, unsigned int *userData* = 0, void * *userPtr* = NULL)

SetCallbacks sets the callback class used by the monitor.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*userCallbacks*	The callbacks class to use. This will be a pointer to one of the user's HFPS↩ MonitorCallbacks-derived class.
in	*userData*	Use this parameter to pass a value to be used in your callback functions.
in	*userPtr*	Use this parameter to pass a pointer to be used in your callback functions.

3.28 HFPSMonitorCallbacks Class Reference

Allows the user to be notified when specific events occur in the HFPSMonitor.

```
#include <HFPSMonitorCallbacks.h>
```

Public Member Functions

- virtual void OnNewFrameRate (double fps, unsigned int userData=0, void *userPtr=0)

  OnNewFrameRate is called when the current frame rate value is updated.

3.28.1 Detailed Description

Allows the user to be notified when specific events occur in the HFPSMonitor.

Make a class inheriting from HFPSMonitorCallbacks and override the callback functions you want to act on.

Use the HFPSMonitor's SetCallbacks function to set this class as the one to be called instead of the default, inactive, one.

3.28.2 Member Function Documentation

3.28.2.1 virtual void HFPSMonitorCallbacks::OnNewFrameRate (double *fps*, unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

OnNewFrameRate is called when the current frame rate value is updated.

Parameters

out	*fps*	The frames per second value.

out	*userData*	The value set in the SetCallback function.
out	*userPtr*	The value set in the SetCallback function.

3.29 HFrameAverage Class Reference

Frame averaging.

`#include <HFrameAverage.h>`

Public Member Functions

- void * AllocateAverageBuffer (eHImageFormat format, int width, int height, int bitDepth)

  Allocate and return a temporary image calculation buffer.This buffer must be allocated before any processing is performed.
- void FreeAverageBuffer (void *buffer)

  Free temporary allocated image calculation buffer.
- bool StandardFrameAverage (int arraySize, HImage **srcImageArray, void *averageBuffer, HImage *dest↩ Image)

  Calculate an image average.
- bool RollingFrameAverage (int arraySize, HImage *newImage, HImage *oldestImage, void *averageBuffer, HImage *destImage)

  It is user responsiblity to keep track of the frames to be added versus removed from the rolling buffer.
- void * AllocateAverageBuffer (eHImageFormat format, int width, int height, int bitDepth)

  Allocate and return a temporary image calculation buffer.This buffer must be allocated before any processing is performed.
- void FreeAverageBuffer (void *buffer)

  Free temporary allocated image calculation buffer.
- bool StandardFrameAverage (int arraySize, HImage **srcImageArray, void *averageBuffer, HImage *dest↩ Image)

  Calculate an image average.
- bool RollingFrameAverage (int arraySize, HImage *newImage, HImage *oldestImage, void *averageBuffer, HImage *destImage)

  It is user responsiblity to keep track of the frames to be added versus removed from the rolling buffer.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()
- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.29.1 Detailed Description

Frame averaging.

Two averaging mode are provided. Standard averaging: a group of frames are averaged together. Returned frame is the averaging result. Roling Average: this will not reduce the incoming / outcomming frame rate. Frames are averaged as a first in / first out scenario: younger frames replacing older one.

3.29.2 Member Function Documentation

3.29.2.1 void HFrameAverage::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.29.2.2 eHErrorCode HFrameAverage::GetLastError ( )

See also

HGrabModule::GetLastError

3.29.2.3 void* HFrameAverage::AllocateAverageBuffer (eHImageFormat *format,* int *width,* int *height,* int *bitDepth*)

Allocate and return a temporary image calculation buffer.This buffer must be allocated before any processing is performed.

Returns

buffer allocated.

Parameters

in	*format*	The image format to be allocated.
in	*width*	The width of the image (in pixel).
in	*height*	The height of the image (in pixel).
in	*bitDepth*	The bit depth of the image.

3.29.2.4 void HFrameAverage::FreeAverageBuffer (void * *buffer*)

Free temporary allocated image calculation buffer.

Parameters

in	*buffer*	the image calculation buffer.

3.29.2.5 bool HFrameAverage::StandardFrameAverage (int *arraySize,* HImage ** *srcImageArray,* void * *averageBuffer,* HImage * *destImage*)

Calculate an image average.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*arraySize*	Number of images in the array.

in	*srcImageArray*	Array of images to be average together.
in	*averageBuffer*	Image calculation buffer.
out	*destImage*	Destination image, containing the averaged image.

3.29.2.6 bool HFrameAverage::RollingFrameAverage (int *arraySize,* HImage * *newImage,* HImage * *oldestImage,* void * *averageBuffer,* HImage * *destImage*)

It is user responsiblity to keep track of the frames to be added versus removed from the rolling buffer.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*arraySize*	Number of images to be rolling averaged.
in	*newImage*	the image to be added to the rolling buffer.
in	*oldestImage*	the image to removed from the rolling buffer.
in	*averageBuffer*	average calculation buffer.
out	*destImage*	Destination image, containing the rolling averaged image.

3.29.2.7 void HFrameAverage::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.29.2.8 eHErrorCode HFrameAverage::GetLastError ( )

See also

HGrabModule::GetLastError

3.29.2.9 void* HFrameAverage::AllocateAverageBuffer (eHImageFormat *format,* int *width,* int *height,* int *bitDepth*)

Allocate and return a temporary image calculation buffer.This buffer must be allocated before any processing is performed.

Returns

buffer allocated.

Parameters

in	*format*	The image format to be allocated.
in	*width*	The width of the image (in pixel).
in	*height*	The height of the image (in pixel).
in	*bitDepth*	The bit depth of the image.

3.29.2.10 void HFrameAverage::FreeAverageBuffer (void * *buffer*)

Free temporary allocated image calculation buffer.

Parameters

in	*buffer*	the image calculation buffer.

3.29.2.11 bool HFrameAverage::StandardFrameAverage (int *arraySize*, HImage ** *srcImageArray*, void * *averageBuffer*, HImage * *destImage*)

Calculate an image average.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*arraySize*	Number of images in the array.
in	*srcImageArray*	Array of images to be average together.
in	*averageBuffer*	Image calculation buffer.
out	*destImage*	Destination image, containing the averaged image.

3.29.2.12 bool HFrameAverage::RollingFrameAverage (int *arraySize*, HImage * *newImage*, HImage * *oldestImage*, void * *averageBuffer*, HImage * *destImage*)

It is user responsiblity to keep track of the frames to be added versus removed from the rolling buffer.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*arraySize*	Number of images to be rolling averaged.
in	*newImage*	the image to be added to the rolling buffer.
in	*oldestImage*	the image to removed from the rolling buffer.
in	*averageBuffer*	average calculation buffer.
out	*destImage*	Destination image, containing the rolling averaged image.

3.30 HGPUDeviceManager Class Reference

Public Member Functions

- bool IsIntelGPUH264Present ()

  Check if CPU is Intel QuickSync class and supports Intel GPU H.264 encoding.
- bool IsIntelGPUHEVCPresent ()

  Check if CPU supports Intel GPU HEVC encoding.
- bool EnableIntelOpenCLEncoding ()

  Enable Intel OPEN-CL JPEG encoding.
- bool IsIntelOpenCLEncodingEnabled ()

  Check if Intel OPEN-CL JPEG encoding is enabled.
- bool DisableIntelOpenCLEncoding ()

  Disable Intel OPEN-CL JPEG encoding.

- bool GetIntelGPUDeviceInfo (LPTSTR name, int *major, int *minor)

  Get Intel GPU information.
- int GetNvidiaGPUDeviceCount ()

  Get NVIDIA GPU count.
- bool GetNvidiaGPUDeviceInfo (int index, LPTSTR name, int *major, int *minor)

  Get NVIDIA GPU information.
- bool IsNvidiaH264Present (int device)

  Check if NVIDIA driver supports GPU H.264 encoding.
- bool IsNvidiaGPUHEVCPresent (int device)

  Check if NVIDIA driver supports GPU HEVC encoding.
- bool EnableNvidiaOpenCLEncoding (int device)

  Enable NVIDIA OPEN-CL JPEG encoding.
- bool IsNvidiaOpenCLEncodingEnabled (int device)

  Check if NVIDIA OPEN-CL JPEG encoding is enabled.
- bool DisableNvidiaOpenCLEncoding (int device)

  Disable NVIDIA OPEN-CL JPEG encoding.
- bool EnableNvidiaCudaEncoding (int device)

  Enable NVIDIA CUDA JPEG encoding.
- bool IsNvidiaCudaEncodingEnabled (int device)

  Check if NVIDIA CUDA JPEG encoding is enabled.
- bool DisableNvidiaCudaEncoding (int device)

  Disable NVIDIA CUDA JPEG encoding.
- int GetAMDGPUDeviceCount ()

  Get AMD GPU count.
- bool GetAMDGPUDeviceInfo (int index, LPTSTR name, int *major, int *minor)

  Get AMD GPU information.
- bool IsAMDH264Present (int device)

  Check if AMD driver supports GPU H.264 encoding.
- bool EnableAMDOpenCLEncoding (int device)

  Enable AMD OpenCL JPEG encoding.
- bool IsAMDOpenCLEncodingEnabled (int device)

  Check if AMD OpenCL JPEG encoding enabled.
- bool DisableAMDOpenCLEncoding (int device)

  Disable AMD OpenCL JPEG encoding.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

Static Public Member Functions

- static void Initialize ()

  Initialize GPU device Manager. Must be called before using any GPU processing.
- static void Uninitialize ()

  Uninitialize GPU device Manager. Make sure to call this function when all GPU processing is done.

3.30.1 Member Function Documentation

3.30.1.1 void HGPUDeviceManager::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.30.1.2 eHErrorCode HGPUDeviceManager::GetLastError ( )

See also

HGrabModule::GetLastError

3.30.1.3 bool HGPUDeviceManager::GetIntelGPUDeviceInfo (LPTSTR *name,* int * *major,* int * *minor*)

Get Intel GPU information.

Parameters

out	*name*	Intel device name.
out	*major*	Major version for OPEN-CL computing capability.
out	*minor*	Minor version for OPEN-CL computing capability.

3.30.1.4 bool HGPUDeviceManager::GetNvidiaGPUDeviceInfo (int *index,* LPTSTR *name,* int * *major,* int * *minor*)

Get NVIDIA GPU information.

Parameters

in	*index*	NVIDIA device to query.
out	*name*	NVIDIA device name.
out	*major*	Major version for CUDA/OPEN-CL computing capability.
out	*minor*	Minor version for CUDA/OPEN-CL computing capability.

3.30.1.5 bool HGPUDeviceManager::EnableNvidiaOpenCLEncoding (int *device*)

Enable NVIDIA OPEN-CL JPEG encoding.

Parameters

in	*device*	NVIDIA device index.

3.30.1.6 bool HGPUDeviceManager::IsNvidiaOpenCLEncodingEnabled (int *device*)

Check if NVIDIA OPEN-CL JPEG encoding is enabled.

Parameters

in	*device*	NVIDIA device index.

3.30.1.7 bool HGPUDeviceManager::DisableNvidiaOpenCLEncoding (int *device*)

Disable NVIDIA OPEN-CL JPEG encoding.

Parameters

in	*device*	NVIDIA device index.

3.30.1.8 bool HGPUDeviceManager::EnableNvidiaCudaEncoding (int *device*)

Enable NVIDIA CUDA JPEG encoding.

Parameters

in	*device*	NVIDIA device index.

3.30.1.9 bool HGPUDeviceManager::IsNvidiaCudaEncodingEnabled (int *device*)

Check if NVIDIA CUDA JPEG encoding is enabled.

Parameters

in	*device*	NVIDIA device index.

3.30.1.10 bool HGPUDeviceManager::DisableNvidiaCudaEncoding (int *device*)

Disable NVIDIA CUDA JPEG encoding.

Parameters

in	*device*	NVIDIA device index.

3.30.1.11 bool HGPUDeviceManager::GetAMDGPUDeviceInfo (int *index*, LPTSTR *name*, int * *major*, int * *minor*)

Get AMD GPU information.

Parameters

in	*index*	AMD device to query.
out	*name*	AMD device name.
out	*major*	Major version for AMD OPEN-CL computing capability.
out	*minor*	Minor version for AMD OPEN-CL computing capability.

3.30.1.12 bool HGPUDeviceManager::EnableAMDOpenCLEncoding (int *device*)

Enable AMD OpenCL JPEG encoding.

Parameters

in	*device*	AMD device index.

3.30.1.13 bool HGPUDeviceManager::IsAMDOpenCLEncodingEnabled (int *device*)

Check if AMD OpenCL JPEG encoding enabled.

Parameters

in	*device*	AMD device index.

3.30.1.14 bool HGPUDeviceManager::DisableAMDOpenCLEncoding (int *device*)

Disable AMD OpenCL JPEG encoding.

Parameters

in	*device*	AMD device index.

3.31 HGrabCallbacks Class Reference

Allows the user to be notified when specific events occur in the HGrabModule.

```
#include <HGrabCallbacks.h>
```

Public Member Functions

- virtual void OnBeforeImageReceived (unsigned int userData=0, void ∗userPtr=0)

  OnBeforeImageReceived is called before the API starts waiting for a new image. Once this callback is done, the grabbing process will enter a waiting state until an image is received. OnImageReceived will then be called.
- virtual void OnImageReceived (HImage ∗image, unsigned int userData=0, void ∗userPtr=0)

  This function is called when an image has just been received from the grabber.
- virtual void OnImageReleased (unsigned int userData=0, void ∗userPtr=0)

  This function is called immediately after a captured image buffer is released, making it available again for the Grab↩ Module.
- virtual void OnBeforeQueuingGrab (unsigned int userData=0, void ∗userPtr=0)

  This function is called immediately before a buffer is queued for grabbing.
- virtual void OnAfterQueuingGrab (unsigned int userData=0, void ∗userPtr=0)

  This function is called after a buffer has been successfully queued for grabbing.
- virtual void OnBeforeStartStreaming (unsigned int userData=0, void ∗userPtr=0)

  This function is called immediately before starting the streaming process.
- virtual void OnAfterStartStreaming (unsigned int userData=0, void ∗userPtr=0)

  This function is called immediately after the streaming process is started.
- virtual void OnBeforeStopStreaming (unsigned int userData=0, void ∗userPtr=0)

  This function is called immediately before stopping the streaming process.
- virtual void OnAfterStopStreaming (unsigned int userData=0, void ∗userPtr=0)

  This function is called immediately after the streaming process is halted.
- virtual void OnNotification (eHGRABBER_NOTIFICATION notification, eHSUGGESTED_ACTIO↩ N suggestedAction)

  This function is called when the grabber has something to say.
- virtual void OnGPSSentencesReceived (TCHAR ∗sentences)

  This function is called when the grabber send a new batch of GPS sentences.

3.31.1 Detailed Description

Allows the user to be notified when specific events occur in the HGrabModule.

Make a class inheriting from HGrabCallbacks and override the callback functions you want to act on. Important : You should avoid calling HGrabModule's functions from the callback function. For instance, StopStream() can not be called in the OnImageReceived() function.

For example :

```
CMyClass : public HGrabCallbacks
{
   private:
      //custom code...

   public:
      virtual void OnImageReceived(HImage* image, unsigned int userData=0,
                                   void* userPtr=0)
   {
      //An image was just received. (stored in the "image" parameter)
         //Now would be a good time to do some processing
         //or maybe save it to disk ?
      .......
       };

       //more custom code...
};
```

Then use the HGrabModule's SetCallbacks function to set this class as the one to be called instead of the default, inactive, one.

3.31.2 Member Function Documentation

3.31.2.1 virtual void HGrabCallbacks::OnBeforeImageReceived (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

OnBeforeImageReceived is called before the API starts waiting for a new image. Once this callback is done, the grabbing process will enter a waiting state until an image is received. OnImageReceived will then be called.

Parameters

out	*userData*	The value set in the SetCallback function.
out	*userPtr*	The value set in the SetCallback function.

3.31.2.2 virtual void HGrabCallbacks::OnImageReceived (HImage * *image*, unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called when an image has just been received from the grabber.

Parameters

out	*image*	The image just received from the grabber, converted in HImage format. The received HImage must not be released, as the buffer it comprises will be reused for later grabs.
out	*userData*	The userData value set in the SetCallback function.
out	*userPtr*	The userPtr value set in the SetCallback function.

3.31.2.3 virtual void HGrabCallbacks::OnImageReleased (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called immediately after a captured image buffer is released, making it available again for the GrabModule.

Parameters

userData	userData [out] The value set in the SetCallback function.
userPtr	userPtr [out] The value set in the SetCallback function.

3.31.2.4 virtual void HGrabCallbacks::OnBeforeQueuingGrab (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called immediately before a buffer is queued for grabbing.

Parameters

out	*userData*	The userData value set in the SetCallback function.
out	*userPtr*	The userPtr value set in the SetCallback function.

3.31.2.5 virtual void HGrabCallbacks::OnAfterQueuingGrab (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called after a buffer has been successfully queued for grabbing.

Parameters

out	*userData*	The userData value set in the SetCallback function.
out	*userPtr*	The userPtr value set in the SetCallback function.

3.31.2.6 virtual void HGrabCallbacks::OnBeforeStartStreaming (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called immediately before starting the streaming process.

Parameters

out	*userData*	The userData value set in the SetCallback function.
out	*userPtr*	The userPtr value set in the SetCallback function.

3.31.2.7 virtual void HGrabCallbacks::OnAfterStartStreaming (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called immediately after the streaming process is started.

Parameters

out	*userData*	The userData value set in the SetCallback function.
out	*userPtr*	The userPtr value set in the SetCallback function.

3.31.2.8 virtual void HGrabCallbacks::OnBeforeStopStreaming (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called immediately before stopping the streaming process.

Parameters

out	*userData*	The userData value set in the SetCallback function.
out	*userPtr*	The userPtr value set in the SetCallback function.

3.31.2.9 virtual void HGrabCallbacks::OnAfterStopStreaming (unsigned int *userData* = 0, void * *userPtr* = 0) [inline],[virtual]

This function is called immediately after the streaming process is halted.

Parameters

out	*userData*	The userData value set in the SetCallback function.
out	*userPtr*	The userPtr value set in the SetCallback function.

3.31.2.10 virtual void HGrabCallbacks::OnNotification (eHGRABBER_NOTIFICATION *notification*, eHSUGGESTED_ACTION *suggestedAction*) [inline],[virtual]

This function is called when the grabber has something to say.

Parameters

notification	Notification from the grabber
suggestedAction	Action suggested by the grabber

3.31.2.11 virtual void HGrabCallbacks::OnGPSSentencesReceived (TCHAR * *sentences*) [inline],[virtual]

This function is called when the grabber send a new batch of GPS sentences.

Parameters

sentences	...

3.32 HGrabModule Class Reference

This class manages image acquisition from a camera.

```
#include <HGrabModule.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
  Set class behavior when encountering an error.
- eHErrorCode GetLastError ()
  Retrieve the error code set by the last function called.

CARD LOADING/UNLOADING FUNCTIONS

- unsigned int GetDriverCount (bool loadableOnly=true)
  Retrieve the number of grabbers that are supported by the current version of the Hermes API.
- LPCTSTR GetDriverInfo (unsigned int index, eHSupportedModule *driverID, bool loadableOnly=true)
  Retrieve the name and grabber identifier of a specific grabber.
- LPCTSTR GetDriverInfo (eHSupportedModule driverID, bool loadableOnly=true)
  Retrieve the name of the grabber by using its identifier.

- bool Load (eHSupportedModule card, LPCTSTR appName, bool showError=false)
  Initialize and setup the module for grabbing, loading the grabber's driver.
- bool Load (eHSupportedModule card, LPCTSTR appName, bool clearSettings, LPCTSTR xmlFilePath, bool showError)
  Initialize and setup the module for grabbing, loading the grabber's driver.
- bool Unload ()
  Unload will uninitialize and close the grabber driver.
- eHSupportedModule GetCurrentGrabber ()
  Retrieve the identifier of the currently loaded grabber.
- bool ExportGrabberSettings (LPCTSTR filePath)
  Saves the settings of the currently loaded grabber to an XML file.

STREAM CONTROL FUNCTIONS

- bool StartStreaming ()
  Start the streaming process, telling the grabber to start capturing images at the maximum possible speed.
- bool StopStreaming ()
  Stop the streaming process.
- bool IsStreaming ()
  Get the streaming state, i.e.

CALLBACKS RELATED FUNCTIONS

- bool SetCallbacks (HGrabCallbacks *userCallbacks, unsigned int userData=0, void *userPtr=NULL)
  Set the callback class used by the grabbing module.

MISC FUNCTIONS

- bool ShowProperties (HWND parentWndHandle)
  Display the property pages of the currently loaded frame grabber.
- HTimeSourceSyncStatus GetTimeSourceSyncStatus ()
  Get time source synchronization status.

GRABBER INFO FUNCTIONS

Some grabber provide various informations about themselves.

Use the following functions to retreive those informations.

- long GetGrabberInfoCount ()
  Get the number of grabber informations.
- bool GetGrabberInfoName (int index, LPTSTR name)
  Get the label of an information.
- bool GetGrabberInfoValueType (LPCTSTR name, eHGrabberInfoType *type)
  Retreive the type of each information.
- bool GetGrabberInfoValueLong (LPCTSTR name, long *value)
  Retreive the information's value for long types.
- bool GetGrabberInfoValueDouble (LPCTSTR name, double *value)
  Retreive the information's value for double types.
- bool GetGrabberInfoValueString (LPCTSTR name, LPTSTR value)
  Retreive the information's value for string types.

GPS FUNCTIONS

- void **EnableGPSCallbackNotification** (bool enable)
- void **EnableSharedGPSUpdate** (bool enable)
- bool **IsGPSCallbackNotification** ()
- bool **IsUpdatingSharedGPS** ()

IMAGE SIZE FUNCTIONS

- bool SetROI (unsigned int offsetX, unsigned int offsetY, unsigned int sizeX, unsigned int sizeY)
  Directly set the Region Of Interest.
- bool GetROI (unsigned int &offsetX, unsigned int &offsetY, unsigned int &sizeX, unsigned int &sizeY)
  Retrieve the position and dimension of the current Region Of Interest (ROI).
- bool SetBinning (unsigned int value)
  Directly set the binning factor of the camera, if supported.
- bool GetBinning (unsigned int &value)
  Retrieve current binning factor of the camera.
- bool GetMaxResolution (unsigned int &maxSizeX, unsigned int &maxSizeY)
  Retrieve the maximum ROI size that can be currently set.

SETTINGS FUNCTIONS

The function HGrabModule::ShowProperties is the official function to access and change the settings of the camera / frame grabber.

However, a few customers have expressed their desire to control these settings without having to use the Show↩ Properties GUI. The following functions were added to provide direct access to the driver adjustments and settings.

The user must be very careful when using these functions, as a lot of the error checking normally done in ShowProperties will not be implemented here. The number of settings available will also vary depending on the camera's streaming state.

As many drivers do not support these functions yet, it is possible to verify grabber support by opening the ShowProperties dialog and looking for tabs called "Adjustments" and "Settings". Everything displayed on the subsequent pages can be controlled with the following functions. The absence of the "Adjustments" or "Settings" pages means that the driver has not yet been updated to support these functions. If this is the case, please contact NorPix to ask for this particular driver to be prioritized in the porting list.

Also note that some of the settings can not be changed while the camera is streaming. For instance, a setting that would the current image format (mono/color/etc) can not be called while the camera is streaming as it would probably crash the driver along with Hermes. As settings differ from one driver to the other, an accurate list of all available settings and capabilities is unlikely . What can be done is better left for the user to experiment. As a general rule, everything in the "Settings" page can not be changed while the camera is streaming and everything in the "Adjustments" page can be changed regardless of the streaming state.

Since only few drivers are currently supporting these functions, ShowProperties should be prioritized as is the safest way to access the adjustments and settings.

Refer to "DemoGrab" for a more complete code sample.

- bool GetSettingsCount (long ∗count)
  Get the number of settings currently available.
- bool GetSettingsCaps (long index, LPTSTR name, long ∗featureID)
  Get the name and the "feature identifier" of each setting.
- bool GetValuesCount (long featureID, long ∗count)
  Get retrieves the number of available choices for a setting.
- bool GetValuesCaps (long featureID, long index, LPTSTR name, long ∗valueID)
  Get the name and value identifier of a selectable choice.
- bool GetCurrentValues (long featureID, long ∗value)
  Get the currently selected choice for a particular setting.
- bool SetCurrentValues (long featureID, long value)
  Select a new choice for a given setting.

ADJUSTMENTS FUNCTIONS

- bool GetAdjustmentCount (long ∗count)
  Get the number of currently available adjustments.
- bool GetAdjustmentCaps (long index, long ∗featureID, LPTSTR name, long ∗min, long ∗max, double ∗d↩ Min, double ∗dMax, BOOL ∗integer, BOOL ∗logSlider, BOOL ∗manual, BOOL ∗automatic, BOOL ∗one↩ Shot)
  Get the name and the "feature identifier" of each adjustment.

- bool GetIntegerAdjustment (long featureID, long *value)
  Get the current value of a particular adjustment.
- bool SetIntegerAdjustment (long featureID, long value)
  Set the value of a particular adjustment.
- bool GetDoubleAdjustment (long featureID, double *value)
  Get the current value of a particular adjustment.
- bool SetDoubleAdjustment (long featureID, double value)
  Set the value of a particular adjustment.
- bool GetAutomaticAdjustment (long featureID, BOOL *automatic)
  Get the mode of the adjustment.
- bool SetAutomaticAdjustment (long featureID, BOOL automatic)
  Set the mode of the adjustment.
- bool OneShotAdjustment (long featureID)
  Triggers a "One-shot" on an adjustment.

CONFIGURATION FILE FUNCTIONS

- bool **UseConfigurationFile** ()
- bool **SetConfigurationFile** ()
- bool **GetConfigurationFile** (LPTSTR filePath)

IMAGE INFORMATION FUNCTIONS

- unsigned int GetImageWidth ()
  Retrieve the width of the images captured by the grabber.
- unsigned int GetImageHeight ()
  Retrieves the height of images captured by the grabber.
- unsigned int GetImageBitDepth ()
  Retrieve the bit depth of images captured by the grabber.
- unsigned int GetImageBitDepthReal ()
  Retrieve the real bit depth of images captured by the grabber.
- unsigned int GetImageSizeBytes ()
  Retrieves the size, in bytes, of each image captured by the grabber.
- eHImageFormat GetImageFormat ()
  Retrieve the format of images captured by the grabber.

TIMESOURCE GPS FUNCTIONS

- bool IsGPSSupported ()
  Check if the selected time source card has GPS capabilities.
- bool GetGPSData (CGPSDataStruct *gpsData, bool readNow=false)
  GetGPSData retrieves the GPS data (latitude, longitude, altitude) from the time source card.
- bool StopReadingGPSData ()
  Stop the GPS data monitoring that was initiated when the GetGPSData function was called.

COM PORT FUNCTIONS

- bool IsCOMPortAvailable ()
  Check availability of the Serial Communication interface.
- bool Transmit (void *pBuffer, unsigned long *pBufferSize, unsigned long serialTimeout)
  Send a buffer to the camera through the COM port.
- bool Receive (void *pBuffer, unsigned long *pBufferSize, unsigned long serialTimeout)
  Retrieves the camera response.
- bool GetEndChar (long *endChar)
  Retrieves the command end character index.
- bool GetTextMode (long *textMode)
  Retrieves the current command format.
- bool GetProgFolder (LPTSTR pathName)
  Retrieves the folder path for program files.

- bool GetSavedProgFolder (LPTSTR pathName)
  Retrieve the last program folder path.
- bool GetCommunicationType (long ∗comType)
  GetCommunicationType retrieves the communication type.

ADVANCED SETTINGS FUNCTIONS

- bool IsSupportAdvanceSettings ()
  Check if the grabber supports advanced settings.
- bool DoModalAdvanceSettingsWindow ()
  Create and show the advanced settings in a modal dialog.
- bool CreateAdvanceSettingsWindow (HWND parent)
  Create a child window to display the advanced settings.
- bool ReSizeAdvanceSettingsWindow (int x, int y, int width, int height)
  Resize the advanced settings window.
- bool ShowAdvanceSettingsWindow (bool show)
  Toggle the visibility of the advanced settings window.
- bool CloseAdvanceSettingsWindow ()
  Close the advanced settings window.
- bool GetCategoriesCount (long ∗count)
  Get the number of categories.
- bool GetCategoriesCap (long index, long ∗cat_id, LPTSTR name)
  Get the name and identifier of each category.
- bool GetFeatureCount (long cat_id, long ∗count)
  Get the feature count in a specific category.
- bool GetFeatureCap (long index, long cat_id, long ∗feature_id, LPTSTR name, eAdvanceSettings↩FeatureType ∗type, eAdvanceSettingsLevel ∗level)
  Get information on a feature.
- bool Accessible (long cat_id, long feature_id, BOOL ∗access)
  Check the feature's current status (read-write or read-only).
- bool GetEnumFeatureValueCount (long cat_id, long feature_id, long ∗count)
  Get how many selectable choices are available for an enum feature.
- bool GetEnumFeatureValueCap (long index, long cat_id, long feature_id, LPTSTR name, long ∗value)
  Get the name and value identifier of a selectable choice.
- bool GetEnumFeatureValue (long cat_id, long feature_id, long ∗value)
  Get the currently selected choice for a feature.
- bool SetEnumFeatureValue (long cat_id, long feature_id, long value)
  Select a new choice for feature.
- bool GetIntegerFeatureCap (long cat_id, long feature_id, long ∗min, long ∗max)
  Get the range (integer) of a feature.
- bool GetIntegerFeatureValue (long cat_id, long feature_id, long ∗value)
  Get the value (integer) of a feature.
- bool SetIntegerFeatureValue (long cat_id, long feature_id, long value)
  Set the value (integer) of a feature.
- bool GetDoubleFeatureCap (long cat_id, long feature_id, double ∗min, double ∗max)
  Get the range (double) of a feature.
- bool GetDoubleFeatureValue (long cat_id, long feature_id, double ∗value)
  Get the value (double) of a feature.
- bool SetDoubleFeatureValue (long cat_id, long feature_id, double value)
  Set the value (double) of a feature.
- bool GetBoolFeatureValue (long cat_id, long feature_id, BOOL ∗value)
  Get the value (bool) of a feature.
- bool SetBoolFeatureValue (long cat_id, long feature_id, BOOL value)
  Set the value (bool) of a feature.
- bool ExecuteCommandFeature (long cat_id, long feature_id)
  Execute a command defined by the command feature.

I/O FUNCTIONS

- HIODriver * GetIODriver ()
  Retrieve a pointer to the grabber IO Driver.

VIDEO OUTPUT FUNCTIONS

- HVideoOutputDriver * **GetVideoOutputDriver** ()

AUDIO FUNCTIONS

- HGrabAudioModule * **GetGrabAudioModule** ()

GRABBER ENUMERATION

Those functions allow to detect any connected cameras that can be used by a specific driver.

IMPORTANT : They can only be called BEFORE any of the Load(...) function.

- bool StartQueryMode (eHSupportedModule driverID)
  Start the Query mode for a specific driver.
- bool GetGrabberCount (long *count)
  Get the number of connected cameras that can use the loaded driver.
- bool GetGrabberName (long index, LPTSTR name)
  Get the unique name of a connected camera.
- bool SetDefaultGrabber (LPCTSTR appName, long index)
  Set the camera that will be loaded by default the next time a Load(...) function is called.
- bool EndQueryMode ()
  Terminate the Query mode.

BUFFERS FUNCTIONS

- void SetBufferCount (unsigned int count)
  Set the number of buffers to be used by the grabbing mechanism.
- unsigned int GetBufferCount ()
  Get the number of buffers used by the grabbing mechanism.
- long GetBufferUsage ()
  Get the number of frames waiting to be processed by the OnImageReceived callback.
- long GetMaxBufferUsage ()
  Get the maximum number of buffers reached since acquisition started (since StartStreaming()).
- long GetTotalBufferCount ()
  Get the total number of frames allocated for capture.

GRABBER NOTIFICATIONS FUNCTIONS

- bool **IsSupportGabberNotification** (eHGRABBER_NOTIFICATION notification)

PAN/ZOOM SUPPORT FUNCTIONS

- bool IsSupportPanTilt ()
  Check if grabber support PanTilt.
- bool IsSupportZoom ()
  Check if grabber support Zoom.
- bool GetPanTiltXRange (double *min, double *max)
  Get PanTilt adjustment range.
- bool GetZoomRange (double *min, double *max)
  Get zoom adjustment range.
- bool SetPanTilt (double x, double y)
- bool SetZoom (double zoom)
- bool GetPanTilt (double *x, double *y)
- bool GetZoom (double *zoom)

3.32.1 Detailed Description

This class manages image acquisition from a camera.

The Hermes grabbing module purpose is to connect to the image capturing hardware and convert the images to a generic image format. (see HImage)

The typical (simplified) call order is :

```
HGrabModule module;       //Instantiates a grab module.
module.SetCallbacks(...);//Prepare the callbacks.
module.Load(..., ...);    //Loads the grabber driver.
module.StartStreaming(); //Starts image capture.
...                       //Processes in callback functions.
module.StopStreaming();  //Stops image capture.
module.Unload();          //Unloads the grabber driver.
```

3.32.2 Member Function Documentation

3.32.2.1 void HGrabModule::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

Set class behavior when encountering an error.

By default, debugString and infoString are enabled and visual is disabled. DebugString and InfoString can be seen trough a program that hooks itself to the system debugger such as Hermes API's NpxDebug.exe

Parameters

in	*debugString*	If true, when an error is detected, the error message will be automatically sent to the System Debugger.
in	*visual*	If true, when an error is detected, the error message will be automatically shown in a standard windows message box. It is recommended to set this to false for public releases.
in	*infoString*	If true, the information string will be sent to the System debugger. Information strings are status information related to the current process. Info strings are not errors.

3.32.2.2 eHErrorCode HGrabModule::GetLastError ( )

Retrieve the error code set by the last function called.

For example, if ModuleX.Foo() returns false, immediately call ModuleX.GetLastError() to get the error code. This error code can be transmitted to the HErrorManager::GetErrorString(eHErrorCode code). The returned value is a string describing the error. This is the same error message shown when the debugString and/or visual parameters of SetErrorOutput() are enabled.

Returns

The error code.

3.32.2.3 unsigned int HGrabModule::GetDriverCount (bool *loadableOnly* = `true`)

Retrieve the number of grabbers that are supported by the current version of the Hermes API.

This function is often used in conjunction with GetDriverInfo to build a list or menu presenting the choices available to the user.

Returns

The number of supported grabbers.

Parameters

in	*loadableOnly*	If true, will only check drivers that can actually be loaded. (Camera driver installed, etc.)

3.32.2.4 LPCTSTR HGrabModule::GetDriverInfo (unsigned int *index*, eHSupportedModule * *driverID*, bool *loadableOnly* = `true`)

Retrieve the name and grabber identifier of a specific grabber.

Returns

The function will return a string holding the grabber name for the specified index. Asking for an out-of-bounds index will return NULL.

Parameters

in	*index*	The index of the grabber to get information on. Valid values range from 0 up to (GetDriverCount() - 1).
out	*driverID*	The identifier of the grabber.
in	*loadableOnly*	If true, will only check drivers that can actually be loaded. (Camera driver installed, etc.)

3.32.2.5 LPCTSTR HGrabModule::GetDriverInfo (eHSupportedModule *driverID*, bool *loadableOnly* = `true`)

Retrieve the name of the grabber by using its identifier.

Returns

The grabber name for the specified index. Asking for an invalid driverID will return NULL.

Parameters

in	*driverID*	The identifier of the grabber.
in	*loadableOnly*	If true, will only check drivers that can actually be loaded. (Camera driver installed, etc.)

3.32.2.6 bool HGrabModule::Load (eHSupportedModule *card*, LPCTSTR *appName*, bool *showError* = `false`)

Initialize and setup the module for grabbing, loading the grabber's driver.

This must be called before starting streaming.

Returns

True if the function is successful or false if the function failed. In case of failure, GetLastError can be called to retrieve the error code.

Parameters

in	*card*	The identifier of the grabber driver that will be managed by the module.
in	*appName*	A string including the name of your application. This name will be used to set a registry path where the grabber settings will be saved. It will have the form " HKEY_CURRENT_USER \ Software \ appName \ Grabman \ ".
in	*showError*	If true, Hermes will show an error message if the grabber fails to initialize properly.

3.32.2.7 bool HGrabModule::Load (eHSupportedModule *card,* LPCTSTR *appName,* bool *clearSettings,* LPCTSTR *xmlFilePath,* bool *showError*)

Initialize and setup the module for grabbing, loading the grabber's driver.

This must be called before starting streaming.

Returns

True if the function is successful or false if the function failed. In case of failure, GetLastError can be called to retrieve the error code.

Parameters

in	*card*	The identifier of the grabber driver that will be managed by the module.
in	*appName*	A string including the name of your application. This name will be used to set a registry path where the grabber settings will be saved. It will have the form " HKEY_CURRENT_USER \ Software \ appName \ Grabman \ ".
in	*clearSettings*	If true, Hermes will reset all the camera settings to their default values.
in	*xmlFilePath*	If not NULL, Hermes will load the grabber using the specified settings file.
in	*showError*	If true, Hermes will show an error message if the grabber fails to initialize properly.

3.32.2.8 bool HGrabModule::Unload ( )

Unload will uninitialize and close the grabber driver.

The grabber driver can not be used after calling Unload, unless it is reloaded by calling Load.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.9 eHSupportedModule HGrabModule::GetCurrentGrabber ( )

Retrieve the identifier of the currently loaded grabber.

Returns

The identifier of the currently loaded grabber. If no grabber is loaded, the returned value will be H_UNKNOWN.

3.32.2.10 bool HGrabModule::ExportGrabberSettings (LPCTSTR *filePath*)

Saves the settings of the currently loaded grabber to an XML file.

That file can then be reused with the Load function.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filePath*	Full file path to the file where the settings will be saved.

3.32.2.11 bool HGrabModule::StartStreaming ( )

Start the streaming process, telling the grabber to start capturing images at the maximum possible speed.

Load must first be called to initialize the frame grabber/camera before you can StartStreaming. Captured images can then be retrieved and processed by using the callback functions provided in the HGrabCallbacks class. While streaming is enabled, more resources will be required from the CPU and computer bandwidth.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.12 bool HGrabModule::StopStreaming ( )

Stop the streaming process.

The driver will tell the grabber to stop the capture loop and queuing images. Once streaming has stopped, callbacks from the grabbing modules will no longer be received. Stopping streaming while unnecessary will free up system resources for other tasks. Don't call this function from the OnImageReceived callback.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.13 bool HGrabModule::IsStreaming ( )

Get the streaming state, i.e.

if the camera is currently capturing images or not.

Returns

True if streaming, false if not.

3.32.2.14 bool HGrabModule::SetCallbacks (HGrabCallbacks * *userCallbacks*, unsigned int *userData* = 0, void * *userPtr* = NULL)

Set the callback class used by the grabbing module.

Example :

```
class MyCallback : public HGrabCallbacks
{
public:
virtual void OnImageReceived(HImage* image, unsigned int userData=0, void* userPtr=0)
{
        printf("image received !");
}
};

HGrabModule GrabModule;
MyCallback callback;
GrabModule.SetCallback(&callback, 0, NULL); //set the callback class the grabber will use
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

See also

HGrabCallbacks

Parameters

in	*userCallbacks*	The callback class to use. This will relate to one of the user HGrabCallbacks-derived classes.
in	*userData*	Use this parameter to pass a value to be used in your callback functions.
in	*userPtr*	Use this parameter to pass a pointer to be used in your callback functions.

3.32.2.15 bool HGrabModule::ShowProperties (HWND *parentWndHandle*)

Display the property pages of the currently loaded frame grabber.

The pages shown will change depending if the frame grabber is currently streaming or not.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*parentWnd↩ Handle*	Handle to the parent window. If NULL, the dialog will not be modal.

3.32.2.16 long HGrabModule::GetGrabberInfoCount ( )

Get the number of grabber informations.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.17 bool HGrabModule::GetGrabberInfoName (int *index*, LPTSTR *name*)

Get the label of an information.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	Index of the information [0,GetGrabberInfoCount()-1].
in,out	*name*	The label. Allocate at least TCHAR[100].

3.32.2.18 bool HGrabModule::GetGrabberInfoValueType (LPCTSTR *name*, eHGrabberInfoType * *type*)

Retreive the type of each information.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*name*	The label of the information.
out	*type*	The type of information (long, double, string). This will tell which GetGrabber↩InfoValueXXX function to call.

3.32.2.19 bool HGrabModule::GetGrabberInfoValueLong (LPCTSTR *name*, long * *value*)

Retreive the information's value for long types.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*name*	The label of the information.
out	*value*	The long value.

3.32.2.20 bool HGrabModule::GetGrabberInfoValueDouble (LPCTSTR *name*, double * *value*)

Retreive the information's value for double types.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*name*	The label of the information.
out	*value*	The double value.

3.32.2.21 bool HGrabModule::GetGrabberInfoValueString (LPCTSTR *name*, LPTSTR *value*)

Retreive the information's value for string types.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*name*	The label of the information.
out	*value*	The string value.

3.32.2.22 bool HGrabModule::SetROI (unsigned int *offsetX,* unsigned int *offsetY,* unsigned int *sizeX,* unsigned int *sizeY*)

Directly set the Region Of Interest.

As the ROI can not be changed while the grabber is streaming, StopStreaming must be called before SetROI. Many grabbers will round the values to match their internal formats. Thus, it is strongly recommended to call GetROI immediately after setting it, to retrieve the exact ROI value set.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*offsetX*	The horizontal offset of the region.
in	*offsetY*	The vertical offset of the region.
in	*sizeX*	The width of the region.
in	*sizeY*	The height of the region.

3.32.2.23 bool HGrabModule::GetROI (unsigned int & *offsetX,* unsigned int & *offsetY,* unsigned int & *sizeX,* unsigned int & *sizeY*)

Retrieve the position and dimension of the current Region Of Interest (ROI).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*offsetX*	The horizontal offset of the region.
out	*offsetY*	The vertical offset of the region.
out	*sizeX*	The width of the region.
out	*sizeY*	The height of the region.

3.32.2.24 bool HGrabModule::SetBinning (unsigned int *value*)

Directly set the binning factor of the camera, if supported.

The binning can not be changed while the grabber is streaming so StopStreaming must be called SetBinning. Call GetBinning immediately after to retrieve the exact binning value that was set.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

value	value [in] The binning factor.

3.32.2.25 bool HGrabModule::GetBinning (unsigned int & *value*)

Retrieve current binning factor of the camera.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*value*	The binning factor.

3.32.2.26 bool HGrabModule::GetMaxResolution (unsigned int & *maxSizeX,* unsigned int & *maxSizeY*)

Retrieve the maximum ROI size that can be currently set.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*maxSizeX*	The maximum allowed width value.
out	*maxSizeY*	The maximum allowed height value.

3.32.2.27 bool HGrabModule::GetSettingsCount (long * *count*)

Get the number of settings currently available.

For a few drivers, this number of settings can change depending on the camera's streaming status.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*count*	The number of settings available.

3.32.2.28 bool HGrabModule::GetSettingsCaps (long *index,* LPTSTR *name,* long * *featureID*)

Get the name and the "feature identifier" of each setting.

This identifier is used by all other functions to refer to a specific setting.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The index of the settings to retrieve. Value must be between 0 and (Settings↩ Count - 1).
out	*name*	The name of the feature. (ex: "Image format" or"Bit Depth") Allocate at least TCHAR[100].
out	*featureID*	The feature identifier.

3.32.2.29 bool HGrabModule::GetValuesCount (long *featureID,* long ∗ *count*)

Get retrieves the number of available choices for a setting.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier of the setting.
out	*count*	The number of available choices.

3.32.2.30 bool HGrabModule::GetValuesCaps (long *featureID,* long *index,* LPTSTR *name,* long ∗ *valueID*)

Get the name and value identifier of a selectable choice.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The identifier of the parent settings.
in	*index*	The index of the value to retrieve. Value must be between 0 and (ValueCount - 1).
out	*name*	The name of the value. Allocate at least TCHAR[100].
out	*valueID*	The value identifier.

3.32.2.31 bool HGrabModule::GetCurrentValues (long *featureID,* long ∗ *value*)

Get the currently selected choice for a particular setting.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier for which to retrieve the current choice.
out	*value*	The value identifier.

3.32.2.32 bool HGrabModule::SetCurrentValues (long *featureID,* long *value*)

Select a new choice for a given setting.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier for which to set a new choice.
in	*value*	The value identifier.

3.32.2.33 bool HGrabModule::GetAdjustmentCount (long * *count*)

Get the number of currently available adjustments.

For a few drivers, this number of settings can change depending on the camera's streaming status.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*count*	The number of adjustments available.

3.32.2.34 bool HGrabModule::GetAdjustmentCaps (long *index,* long * *featureID,* LPTSTR *name,* long * *min,* long * *max,* double * *dMin,* double * *dMax,* BOOL * *integer,* BOOL * *logSlider,* BOOL * *manual,* BOOL * *automatic,* BOOL * *oneShot*)

Get the name and the "feature identifier" of each adjustment.

The identifier is used in all other functions to reference the adjustment interacted with.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The index of the adjustment to retrieve Value must be between 0 and (AdjustmentsCount - 1).
out	*featureID*	The feature identifier.
out	*name*	The name of the feature. (ex: "Gain" or "Exposure") Allocate at least TCHA↩R[100].
out	*min*	If the adjustment accepts integer values, this is the minimum value accepted.
out	*max*	If the adjustment accepts integer values, this is the maximum value accepted.
out	*dMin*	If the adjustment accepts double values, this is the minimum value accepted.
out	*dMax*	If the adjustment accepts double values, this is the maximum value accepted.
out	*integer*	If TRUE, the adjustment accepts integer values. If FALSE it accepts double values.
out	*logSlider*	TRUE if the use of a logarithmic slider is recommended to set the value. (This is often the case with exposure time)

out	*manual*	TRUE if the value can be changed by the user using SetIntegerAdjustment or SetDoubleAdjustment. If FALSE, the user can not specify a value.
out	*automatic*	TRUE if the adjustment has an automatic mode.
out	*oneShot*	TRUE if the adjustment has a One-Shot capability.

3.32.2.35 bool HGrabModule::GetIntegerAdjustment (long *featureID,* long ∗ *value*)

Get the current value of a particular adjustment.

The adjustment must accept integer values for this function to work.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier of the adjustment.
out	*value*	The current value.

3.32.2.36 bool HGrabModule::SetIntegerAdjustment (long *featureID,* long *value*)

Set the value of a particular adjustment.

The adjustment must accept integer values for this function to work. Only use if "manual" from GetAdjustmentCaps is TRUE.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier of the adjustment.
in	*value*	The new value. Must be between [min, max] read from GetAdjustmentCaps.

3.32.2.37 bool HGrabModule::GetDoubleAdjustment (long *featureID,* double ∗ *value*)

Get the current value of a particular adjustment.

The adjustment must accept double values for this function to work.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier of the adjustment.
out	*value*	The current value.

3.32.2.38 bool HGrabModule::SetDoubleAdjustment (long *featureID,* double *value*)

Set the value of a particular adjustment.

The adjustment must accept double values for this function to work. Only use if "manual" from GetAdjustmentCaps is TRUE.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier of the adjustment.
in	*value*	The new value. Must be between [dMin, dMax] read from GetAdjustmentCaps.

3.32.2.39 bool HGrabModule::GetAutomaticAdjustment (long *featureID,* BOOL * *automatic*)

Get the mode of the adjustment.

When in automatic mode, the value of the adjustment can change during capture without notifying the user. While in automatic mode, you can not change the current value by a call to SetXXXAdjustment. Only use if "automatic" from GetAdjustmentCaps is TRUE.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier of the adjustment.
out	*automatic*	If TRUE, the adjustment is currently in automatic mode.

3.32.2.40 bool HGrabModule::SetAutomaticAdjustment (long *featureID,* BOOL *automatic*)

Set the mode of the adjustment.

Only use if "automatic" from GetAdjustmentCaps is TRUE.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature identifier of the adjustment
in	*automatic*	TRUE to enable automatic mode, FALSE to deactivate the automatic mode

3.32.2.41 bool HGrabModule::OneShotAdjustment (long *featureID*)

Triggers a "One-shot" on an adjustment.

One instance of a "One-Shot" function is a function that adjusts the camera exposure using the next image received, making no further exposure change to the next images. Only use if "oneShot" from GetAdjustmentCaps is TRUE.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*featureID*	The feature ID

3.32.2.42 unsigned int HGrabModule::GetImageWidth ( )

Retrieve the width of the images captured by the grabber.

Returns

This image width of the captured images. If no grabber is selected or if the value can not be read from the hardware, the value will be 0.

3.32.2.43 unsigned int HGrabModule::GetImageHeight ( )

Retrieves the height of images captured by the grabber.

Returns

The image height of the captured images. If no grabber is selected or if the value can not be read from the hardware, the value will be 0.

3.32.2.44 unsigned int HGrabModule::GetImageBitDepth ( )

Retrieve the bit depth of images captured by the grabber.

Returns

The bit depth of the captured images. If no grabber is selected or if the value can not be read from the hardware, the value will be 0.

3.32.2.45 unsigned int HGrabModule::GetImageBitDepthReal ( )

Retrieve the real bit depth of images captured by the grabber.

Returns

The real bit depth of captured images. If no grabber is selected or if the value can not be read from the hardware, the returned value will be 0.

3.32.2.46 unsigned int HGrabModule::GetImageSizeBytes ( )

Retrieves the size, in bytes, of each image captured by the grabber.

Returns

The size of captured images in bytes. If no grabber is selected or if the value can not be read from the hardware, the returned value is 0.

3.32.2.47 eHImageFormat HGrabModule::GetImageFormat ( )

Retrieve the format of images captured by the grabber.

Returns

The format of the captured images. If no grabber is selected or if the format can not be determined, the returned value will be H_IMAGE_UNKNOWN.

3.32.2.48 bool HGrabModule::IsGPSSupported ( )

Check if the selected time source card has GPS capabilities.

Returns

True if the currently selected time source card has GPS capabilities or false otherwise.

3.32.2.49 bool HGrabModule::GetGPSData (CGPSDataStruct * *gpsData*, bool *readNow* = `false`)

GetGPSData retrieves the GPS data (latitude, longitude, altitude) from the time source card.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

`in,out`	*gpsData*	A valid pointer to a CGPSDataStruct object has to be passed. After a successful call, this parameter will hold the GPS data that was retrieved.
`in`	*readNow*	When set to true, the GPS data is retrieved right away. If set to false(default), the first call to this function will start a thread that updates the GPS data every 250ms.

3.32.2.50 bool HGrabModule::StopReadingGPSData ( )

Stop the GPS data monitoring that was initiated when the GetGPSData function was called.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.51 bool HGrabModule::IsCOMPortAvailable ( )

Check availability of the Serial Communication interface.

This interface is used with frame grabbers to provide access to asynchronous serial reading and writing functions in the camera's API, via COM port or a CLSER***.dll, where *** is specific to the frame grabber vendor.

Returns

True if Serial Communication interface is available or false otherwise.

3.32.2.52 bool HGrabModule::Transmit (void * *pBuffer,* unsigned long * *pBufferSize,* unsigned long *serialTimeout*)

Send a buffer to the camera through the COM port.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pBuffer*	The buffer to be transmitted.
in,out	*pBufferSize*	The buffer size, indicating the maximum number of bytes to be written. Upon a successful call, bufferSize contains the number of bytes written.
in	*serialTimeout*	The timeout in milliseconds.

3.32.2.53 bool HGrabModule::Receive (void * *pBuffer,* unsigned long * *pBufferSize,* unsigned long *serialTimeout*)

Retrieves the camera response.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*pBuffer*	Points to a user-allocated buffer. Upon a successful call, buffer contains the data read from the serial device. Caller should ensure that buffer is at least bufferSize in size.
in,out	*pBufferSize*	Contains the buffer size indicating the maximum number of bytes that the buffer can accommodate. Upon a successful call, bufferSize contains the number of bytes read successfully from the camera.
in	*serialTimeout*	Indicates the timeout in milliseconds.

3.32.2.54 bool HGrabModule::GetEndChar (long * *endChar*)

Retrieves the command end character index.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*endChar*	The end character index. • 0 - NULL character • 1 - < CR > • 2 - < LF > • 3 - < CR LF >

3.32.2.55 bool HGrabModule::GetTextMode (long * *textMode*)

Retrieves the current command format.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*textMode*	The command format. • 0 : ASCII • 1 : Hexadecimal

3.32.2.56 bool HGrabModule::GetProgFolder (LPTSTR *pathName*)

Retrieves the folder path for program files.

As the user can write scripting files containing commands, he can select the specific folder where these files will be located.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*pathName*	The user defined folder path.

3.32.2.57 bool HGrabModule::GetSavedProgFolder (LPTSTR *pathName*)

Retrieve the last program folder path.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*pathName*	The last program folder path.

3.32.2.58 bool HGrabModule::GetCommunicationType (long * *comType*)

GetCommunicationType retrieves the communication type.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*comType*	The communication type. • 0 : Communication is disabled • 1 : via camera's API • 2 : via CLSER∗∗∗.dll • 3 : via COM Port

3.32.2.59 bool HGrabModule::IsSupportAdvanceSettings ( )

Check if the grabber supports advanced settings.

Returns

True if the grabber supports advanced settings.

3.32.2.60 bool HGrabModule::DoModalAdvanceSettingsWindow ( )

Create and show the advanced settings in a modal dialog.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.61 bool HGrabModule::CreateAdvanceSettingsWindow (HWND *parent*)

Create a child window to display the advanced settings.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*parent*	Handle of the parent window

3.32.2.62 bool HGrabModule::ReSizeAdvanceSettingsWindow (int *x*, int *y*, int *width*, int *height*)

Resize the advanced settings window.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

See also

CreateAdvanceSettingsWindow

Parameters

in	*x*	x offset of the window relative to the left of the desktop
in	*y*	y offset of the window relative to the top of the desktop
in	*width*	Width of the window
in	*height*	Height of the window

3.32.2.63 bool HGrabModule::ShowAdvanceSettingsWindow (bool *show*)

Toggle the visibility of the advanced settings window.

The window must have been created previously by a call to CreateAdvanceSettingsWindow.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

See also

CreateAdvanceSettingsWindow

Parameters

in	*show*	True to show or false to hide the window

3.32.2.64 bool HGrabModule::CloseAdvanceSettingsWindow ( )

Close the advanced settings window.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

See also

CreateAdvanceSettingsWindow

3.32.2.65 bool HGrabModule::GetCategoriesCount (long * *count*)

Get the number of categories.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*count*	The number of supported categories

3.32.2.66 bool HGrabModule::GetCategoriesCap (long *index*, long * *cat_id*, LPTSTR *name*)

Get the name and identifier of each category.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The index of the category to retrieve. Value must be between 0 and (GetCategoriesCount() - 1)
out	*cat_id*	The category identifier
in,out	*name*	The name of the category. Allocate at least "TCHAR[100]".

3.32.2.67 bool HGrabModule::GetFeatureCount (long *cat_id*, long * *count*)

Get the feature count in a specific category.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
out	*count*	The number of features in that category

3.32.2.68 bool HGrabModule::GetFeatureCap (long *index*, long *cat_id*, long * *feature_id*, LPTSTR *name*, eAdvanceSettingsFeatureType * *type*, eAdvanceSettingsLevel * *level*)

Get information on a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The index of the feature to retrieve. Value must be between 0 and (GetFeatureCount() - 1)
in	*cat_id*	The category identifier
out	*feature_id*	The feature identifier
in,out	*name*	The feature name. Allocate at least "TCHAR[100]".
out	*type*	The feature type (enumeration, tnteger, double, bool and command)
out	*level*	The feature recommended visibility level (Beginner, Expert and Guru)

3.32.2.69 bool HGrabModule::Accessible (long *cat_id*, long *feature_id*, BOOL * *access*)

Check the feature's current status (read-write or read-only).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
out	*access*	TRUE if read-write, FALSE if read-only

3.32.2.70 bool HGrabModule::GetEnumFeatureValueCount (long *cat_id,* long *feature_id,* long * *count*)

Get how many selectable choices are available for an enum feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
out	*count*	The number of available choices

3.32.2.71 bool HGrabModule::GetEnumFeatureValueCap (long *index,* long *cat_id,* long *feature_id,* LPTSTR *name,* long * *value*)

Get the name and value identifier of a selectable choice.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The index of the choice to retrieve for the specified category/feature
in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
in,out	*name*	The name of the value. Allocate at least "TCHAR[100]".
out	*value*	The value identifier

3.32.2.72 bool HGrabModule::GetEnumFeatureValue (long *cat_id,* long *feature_id,* long * *value*)

Get the currently selected choice for a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier

out	*value*	The value identifier

3.32.2.73 bool HGrabModule::SetEnumFeatureValue (long *cat_id,* long *feature_id,* long *value*)

Select a new choice for feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
in	*value*	The value identifier

3.32.2.74 bool HGrabModule::GetIntegerFeatureCap (long *cat_id,* long *feature_id,* long * *min,* long * *max*)

Get the range (integer) of a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
out	*min*	The feature min value
out	*max*	The feature max value

3.32.2.75 bool HGrabModule::GetIntegerFeatureValue (long *cat_id,* long *feature_id,* long * *value*)

Get the value (integer) of a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
out	*value*	The current value

3.32.2.76 bool HGrabModule::SetIntegerFeatureValue (long *cat_id,* long *feature_id,* long *value*)

Set the value (integer) of a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
in	*value*	The value

3.32.2.77 bool HGrabModule::GetDoubleFeatureCap (long *cat_id,* long *feature_id,* double * *min,* double * *max*)

Get the range (double) of a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
out	*min*	The feature min value
out	*max*	The feature max value

3.32.2.78 bool HGrabModule::GetDoubleFeatureValue (long *cat_id,* long *feature_id,* double * *value*)

Get the value (double) of a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
out	*value*	The current value

3.32.2.79 bool HGrabModule::SetDoubleFeatureValue (long *cat_id,* long *feature_id,* double *value*)

Set the value (double) of a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
in	*value*	The value

3.32.2.80 bool HGrabModule::GetBoolFeatureValue (long *cat_id,* long *feature_id,* BOOL * *value*)

Get the value (bool) of a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
out	*value*	The current value

3.32.2.81 bool HGrabModule::SetBoolFeatureValue (long *cat_id,* long *feature_id,* BOOL *value*)

Set the value (bool) of a feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier
in	*value*	The value

3.32.2.82 bool HGrabModule::ExecuteCommandFeature (long *cat_id,* long *feature_id*)

Execute a command defined by the command feature.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*cat_id*	The category identifier
in	*feature_id*	The feature identifier

3.32.2.83 HIODriver* HGrabModule::GetIODriver ( )

Retrieve a pointer to the grabber IO Driver.

Check HIODriver for details on how to use that class.

Returns

The grabber IO driver or NULL if none.

3.32.2.84 bool HGrabModule::StartQueryMode (eHSupportedModule *driverID*)

Start the Query mode for a specific driver.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*driverID*	The identifier of the driver.

3.32.2.85 bool HGrabModule::GetGrabberCount (long * *count*)

Get the number of connected cameras that can use the loaded driver.

The Query mode must be active (through a call to StartQueryMode) for this function to work.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*count*	The camera count.

3.32.2.86 bool HGrabModule::GetGrabberName (long *index,* LPTSTR *name*)

Get the unique name of a connected camera.

The Query mode must be active (through a call to StartQueryMode) for this function to work.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The camera index [0, GrabberCount-1].
in,out	*name*	The camera name. Allocate a TCHAR[250].

3.32.2.87 bool HGrabModule::SetDefaultGrabber (LPCTSTR *appName,* long *index*)

Set the camera that will be loaded by default the next time a Load(...) function is called.

The Query mode must be active (through a call to StartQueryMode) for this function to work.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*appName*	A string including the name of your application. This name will be used to set a registry path where the grabber settings will be saved. It will have the form " HKEY_CURRENT_USER \ Software \ appName \ Grabman \ ".

in	*index*	The camera index [0, GrabberCount-1].

3.32.2.88 bool HGrabModule::EndQueryMode ( )

Terminate the Query mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.89 void HGrabModule::SetBufferCount (unsigned int *count*)

Set the number of buffers to be used by the grabbing mechanism.

Having a large number of buffers will help to prevent frame drops. However, having too many buffers will quickly eat the available RAM, so there is a compromise to be made. The default 5 buffers should be enough for most applications. SetBufferCount must be called before loading the frame grabber, as the buffer pool is created in the Load function.

Parameters

in	*count*	The number of buffers to create.

3.32.2.90 unsigned int HGrabModule::GetBufferCount ( )

Get the number of buffers used by the grabbing mechanism.

The default value is 5, but can be changed with the SetBufferCount function.

Returns

The number of buffers used by the grabbing mechanism.

3.32.2.91 long HGrabModule::GetBufferUsage ( )

Get the number of frames waiting to be processed by the OnImageReceived callback.

If this number reaches the total buffer count, the application will start dropping frames. The best place to call this function is in the OnImageReceived callback function.

Returns

The number of buffers waiting to be processed.

3.32.2.92 long HGrabModule::GetMaxBufferUsage ( )

Get the maximum number of buffers reached since acquisition started (since StartStreaming()).

Returns

The maximum buffer usage since acquisition started.

3.32.2.93 long HGrabModule::GetTotalBufferCount ( )

Get the total number of frames allocated for capture.

If there was enough resources to allocate all buffers, this value should be equal to what was set in SetBufferCount.

Returns

The total number of frames allocated for acquisition.

3.32.2.94 bool HGrabModule::IsSupportPanTilt ( )

Check if grabber support PanTilt.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.95 bool HGrabModule::IsSupportZoom ( )

Check if grabber support Zoom.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.32.2.96 bool HGrabModule::GetPanTiltXRange (double * *min*, double * *max*)

Get PanTilt adjustment range.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*min*	minmum value
out	*max*	maxmum value

3.32.2.97 bool HGrabModule::GetZoomRange (double * *min*, double * *max*)

Get zoom adjustment range.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*min*	minmum value
out	*max*	maxmum value

3.32.2.98 bool HGrabModule::SetPanTilt (double *x*, double *y*)

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*x*	Pan value to adjust
in	*y*	Title value to adjust

3.32.2.99 bool HGrabModule::SetZoom (double *zoom*)

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*zoom*	zoom value to adjust

3.32.2.100 bool HGrabModule::GetPanTilt (double * *x*, double * *y*)

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*x*	Pan value to adjust
out	*y*	Title value to adjust

3.32.2.101 bool HGrabModule::GetZoom (double * *zoom*)

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*zoom*	zoom value to adjust

3.33 HGrayConverter Class Reference

This class converts an HImage to grey monochrome 8 bit format.

```
#include <HGrayConverter.h>
```

Public Member Functions

- bool GrayToMono8 (int offsetX, int offsetY, int sizeX, int sizeY, HImage *srcImage, HImage *destImage)
  Convert an image to the grey monochrome 8 bit format.
- bool CheckGrayConvertionImageFormat (eHImageFormat format, int bitDepth)
  Check if the image format and bit depth are supported by the image conversion routine.
- bool GrayToMono8 (int offsetX, int offsetY, int sizeX, int sizeY, HImage *srcImage, HImage *destImage)
  Convert an image to the grey monochrome 8 bit format.
- bool CheckGrayConvertionImageFormat (eHImageFormat format, int bitDepth)
  Check if the image format and bit depth are supported by the image conversion routine.

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.33.1 Detailed Description

This class converts an HImage to grey monochrome 8 bit format.

The color conversion formulae is the following: Grey = 0.299*R + 0.587*G + 0.114*B.

3.33.2 Member Function Documentation

3.33.2.1 void HGrayConverter::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.33.2.2 eHErrorCode HGrayConverter::GetLastError ( )

See also

HGrabModule::GetLastError

3.33.2.3 bool HGrayConverter::GrayToMono8 (int *offsetX*, int *offsetY*, int *sizeX*, int *sizeY*, HImage * *srcImage*, HImage * *destImage*)

Convert an image to the grey monochrome 8 bit format.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*offsetX*	The horizontal offset of the region to be converted.
in	*offsetY*	The vertical offset of the region to be converted.
in	*sizeX*	The width of the region to be converted.
in	*sizeY*	The height of the region to be converted.
in	*srcImage*	Source image to be converted
out	*destImage*	Destination for the converted image

3.33.2.4 bool HGrayConverter::CheckGrayConvertionImageFormat (eHImageFormat *format*, int *bitDepth*)

Check if the image format and bit depth are supported by the image conversion routine.

Returns

True if the image format and bit depth combination is supported. False otherwise.

Parameters

in	*format*	Source image format.
in	*bitDepth*	Source image bit depth.

3.33.2.5 void HGrayConverter::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.33.2.6 eHErrorCode HGrayConverter::GetLastError ( )

See also

HGrabModule::GetLastError

3.33.2.7 bool HGrayConverter::GrayToMono8 (int *offsetX*, int *offsetY*, int *sizeX*, int *sizeY*, HImage * *srcImage*, HImage * *destImage*)

Convert an image to the grey monochrome 8 bit format.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*offsetX*	The horizontal offset of the region to be converted.
in	*offsetY*	The vertical offset of the region to be converted.
in	*sizeX*	The width of the region to be converted.
in	*sizeY*	The height of the region to be converted.
in	*srcImage*	Source image to be converted
out	*destImage*	Destination for the converted image

3.33.2.8 bool HGrayConverter::CheckGrayConvertionImageFormat (eHImageFormat *format*, int *bitDepth*)

Check if the image format and bit depth are supported by the image conversion routine.

Returns

True if the image format and bit depth combination is supported. False otherwise.

Parameters

in	*format*	Source image format.
in	*bitDepth*	Source image bit depth.

3.34 HImage Class Reference

The image container used everywhere in the API.

`#include <HImage.h>`

Public Member Functions

- void ReallocateImage (unsigned int width, unsigned int height, unsigned int bitDepth, unsigned int bitDepth↩ Real, eHImageFormat imageFormat)

  Delete the image currently stored in the HImage and will reallocate memory to fit the new image following the specified parameters.
- void ReallocateCompressedImage (eHCompression compression, int compressedImageSizeBytes)

  Delete the compressed data currently stored in the HImage and will reallocate memory to fit the new compressed buffer following the specified parameters.
- bool HasImageDecoded ()

  Check if image has been decoded (provided the image is H264 frame).
- eHCompression GetCompressionFormat ()

  Get compression type.
- int GetCompressedChannelSizeBytes ()

  Get compression buffer size.
- void ∗ GetCompressedChannel ()

  Get compression buffer pointer.
- int GetCompressedImageSizeBytes ()

  Get real compression data size.
- void ∗ GetCompressedImage ()

  Get real compression data pointer.
- void FreeCompressedChannel ()

  Free compression buffer.

IMAGE INFORMATION GETTERS

- unsigned int GetImageWidth ()

  Retrieve the width (in pixels) of the image.
- unsigned int GetImageHeight ()

  Retrieve the height (in pixels) of the image.
- unsigned int GetImageBitDepth ()

  Retrieve the bit depth of the image.
- unsigned int GetImageBitDepthReal ()

  Retrieve the real bit depth of each image channel.
- unsigned int GetImageSizeBytes ()

  Retrieve the size of the image in bytes.
- eHImageFormat GetImageFormat ()

  Retrieves the format of the image.

IMAGE DATA ACCESS

- void ∗ GetRawImageData ()

Retrieve a pointer to the actual image data, as grabbed from the grabber or read from a sequence.

- void * GetRedChannel ()

  Retrieve a pointer to the red image data of a planar image.
- void * GetGreenChannel ()

  Retrieve a pointer to the green image data of a planar image.
- void * GetBlueChannel ()

  Retrieve a pointer to the blue image data of a planar image.

TIMESTAMP RELATED FUNCTIONS

- long GetTimestamp ()

  Retrieve the timestamp of the image in time_t format (32-bit).
- unsigned short GetTimestampMS ()

  Retrieve the milliseconds part of the absolute image timestamp.
- unsigned short GetTimestampUS ()

  Retreive the microseconds part of the absolute image timestamp.
- void SetTimestamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS=0)

  Change the timestamp of a HImage.

METADATA FUNCTIONS

- int GetMetadataCount ()

  Retrieve the number of metadata attached to this image.
- HMetadata * GetMetadata (int index)

  Get one of the metadata attached to the image.
- int GetSpecificMetadataCount (unsigned int uID)

  Retrieve the number of metadata of a specific type attached to this image.
- HMetadata * GetSpecificMetadata (unsigned int uID, int index)

  Get one of the metadata of a specific type attached to the image.
- void AddMetadata (const HMetadata &metadata)

  Add metadata to the image.
- void RemoveAllMetadata ()

  Remove all the metadatas attached to the image, if any.

Static Public Member Functions

STATIC FUNCTIONS

Use those two function to manually create, duplicate and delete HImages.

- static HImage * CreateHImage ()

  Create an empty HImage in which one image from a sequence file can be read.
- static HImage * CloneHImage (HImage *srcImage)

  Create a perfect copy of a source HImage.
- static void CloneHImage (HImage *srcImage, HImage *destImage)

  Create a perfect copy of the srcImage in the destImage.
- static void DeleteHImage (HImage *image)

  Delete an HImage that was allocated with a call to CreateHImage().

3.34.1 Detailed Description

The image container used everywhere in the API.

It comprises the data of a single image and offers several functions to query the image format. Some modules create HImage as their outputs, like HGrabModule (images captured from the camera) or HSequencer (images read from a sequence during browsing or playback). Other classes, like HViewer, use HImages as inputs.

COMPRESSED CHANNEL EXPLANATION

```
ImageData.CompressedChannel = [SIZE][COMPRESSED IMAGE DATA][TIMESTAMP]
Size :                        [4 b.][X bytes              ][8 bytes  ]
Example :                     [1004][image data=1000 bytes][timestamp]

GetCompressedChannel() returns a pointer to the begining of [SIZE]
GetCompressedImage() returns a pointer to the begining of [COMPRESSED IMAGE DATA]
GetCompressedChannelSizeBytes() returns the size of [SIZE]+[COMPRESSED IMAGE
      DATA] (ex : 1004)
GetCompressedImageSizeBytes() returns the size of [COMPRESSED IMAGE DATA] (ex :
      1000)
```

3.34.2 Member Function Documentation

3.34.2.1 static HImage* HImage::CreateHImage ( ) [static]

Create an empty HImage in which one image from a sequence file can be read.

When it is no longer needed, free it by using DeleteHimage.

Returns

A pointer to a newly created HImage.

3.34.2.2 static HImage* HImage::CloneHImage (HImage * *srcImage*) [static]

Create a perfect copy of a source HImage.

When it is no longer needed, free it by using DeleteHimage.

Returns

A pointer to the newly created HImage.

Parameters

in	*srcImage*	The source image to clone.

3.34.2.3 static void HImage::CloneHImage (HImage * *srcImage*, HImage * *destImage*) [static]

Create a perfect copy of the srcImage in the destImage.

For the destImage, only use an HImage you created using HImage::CreateImage. If there is currently any image or data in the destImage, it will be deleted prior to the copy.

Parameters

in	*srcImage*	The source image to clone.
in	*destImage*	The destination image.

3.34.2.4 static void HImage::DeleteHImage (HImage * *image*) [static]

Delete an HImage that was allocated with a call to CreateHImage().

Only user-created HImages should be deleted.

Parameters

in	*image*	The HImage to delete.

3.34.2.5 unsigned int HImage::GetImageWidth ( )

Retrieve the width (in pixels) of the image.

Returns

The width of the image in pixels.

3.34.2.6 unsigned int HImage::GetImageHeight ( )

Retrieve the height (in pixels) of the image.

Returns

The height of the image in pixels.

3.34.2.7 unsigned int HImage::GetImageBitDepth ( )

Retrieve the bit depth of the image.

For example, 16 bits will be needed to store one pixel of a 12-bit image, so the bit depth would be 16.

Returns

The function will return the bit depth of the image. If the HImage is empty, the returned value is 0.

3.34.2.8 unsigned int HImage::GetImageBitDepthReal ( )

Retrieve the real bit depth of each image channel.

For example, a 12-bit mono image will have a bit depth of 16 and a real bit depth of 12. For a 24-bit BGR image, the bit depth would be 24 and the real bit depth would be 8 (because 8 bits per channels).

Returns

The function will return the real bit depth of the images. If the HImage is empty, the returned value is 0.

3.34.2.9 unsigned int HImage::GetImageSizeBytes ( )

Retrieve the size of the image in bytes.

Returns

The size of the image in bytes. If the HImage is empty, the returned value is 0.

3.34.2.10 eHImageFormat HImage::GetImageFormat ( )

Retrieves the format of the image.

Returns

The format of the image.

3.34.2.11 void* HImage::GetRawImageData ( )

Retrieve a pointer to the actual image data, as grabbed from the grabber or read from a sequence.

For monochrome or color packed images, this points to the full image data. In the case of a planar color image (format is H_IMAGE_PLANAR), use the GetRedChannel(), GetGreenChannel() and GetBlueChannel() functions to access the individual color channels.

Returns

A pointer to the image data. If the HImage is empty, the returned value is NULL.

3.34.2.12 void* HImage::GetRedChannel ( )

Retrieve a pointer to the red image data of a planar image.

You can also use GetRawImageData() to access the whole image data (red/green/blue).

Returns

A pointer to the red channel for planar images. If the HImage is empty, the returned value is NULL.

3.34.2.13 void* HImage::GetGreenChannel ( )

Retrieve a pointer to the green image data of a planar image.

If the image format is not planar, the green channel is empty.

Returns

A pointer to the green channel. If the HImage is empty or if the image is not planar, the returned value is NULL.

3.34.2.14 void* HImage::GetBlueChannel ( )

Retrieve a pointer to the blue image data of a planar image.

If the image format is not planar, the blue channel is empty.

Returns

A pointer to the blue channel. If the HImage is empty or if the image is not planar, the returned value is NULL.

3.34.2.15 void HImage::ReallocateImage (unsigned int *width*, unsigned int *height*, unsigned int *bitDepth*, unsigned int *bitDepthReal*, eHImageFormat *imageFormat*)

Delete the image currently stored in the HImage and will reallocate memory to fit the new image following the specified parameters.

WARNING : Don't reallocate an image that you did not create with CreateHImage(). Don't forget to set the timestamp for this new image (if needed).

Parameters

in	*width*	The width of the image (in pixel).
in	*height*	The height of the image (in pixel).
in	*bitDepth*	The bit depth of the image.
in	*bitDepthReal*	The real bit depth of the image.
in	*imageFormat*	The image format.

3.34.2.16 void HImage::ReallocateCompressedImage (eHCompression *compression*, int *compressedImageSizeBytes*)

Delete the compressed data currently stored in the HImage and will reallocate memory to fit the new compressed buffer following the specified parameters.

Parameters

in	*compression*	Compression type
in	*compressed↩ ImageSizeBytes*	Compression buffer size

3.34.2.17 bool HImage::HasImageDecoded ( )

Check if image has been decoded (provided the image is H264 frame).

Returns

True if the image has been decoded, false otherwise.

3.34.2.18 long HImage::GetTimestamp ( )

Retrieve the timestamp of the image in time_t format (32-bit).

The timestamp indicates the exact time at which the image was received from the camera. More precision can be obtained by calling GetTimestampMS/US.

Example :

```
//Print the timestamp of the HImage "image"
long t = image->GetTimestamp();
printf("timestamp : %.19s.%.003hu", _tctime(&t), image->GetTimestampMS());
```

Returns

The absolute timestamp in seconds.

3.34.2.19 unsigned short HImage::GetTimestampMS ( )

Retrieve the milliseconds part of the absolute image timestamp.

Depending on the latency of the operating system, the precision might be off by 1 to 3 milliseconds.

Returns

The milliseconds part of the image timestamp.

3.34.2.20 unsigned short HImage::GetTimestampUS ( )

Retreive the microseconds part of the absolute image timestamp.

This value will be 0 unless the frame was timestamped with an external high performance timer device used in conjunction with the grabber.

Returns

The microsecond part of the image timestamp.

3.34.2.21 void HImage::SetTimestamp (long *timeStamp*, unsigned short *timeStampMS*, unsigned short *timeStampUS* = 0)

Change the timestamp of a HImage.

Parameters

in	*timeStamp*	The absolute timestamp in seconds (time_t).
in	*timeStampMS*	The milliseconds part of the image timestamp.
in	*timeStampUS*	The microseconds part of the image timestamp.

3.34.2.22 int HImage::GetMetadataCount ( )

Retrieve the number of metadata attached to this image.

Returns

The metadata count.

3.34.2.23 HMetadata* HImage::GetMetadata (int *index*)

Get one of the metadata attached to the image.

Returns

The metadata.

Parameters

in	*index*	The index of the metadata,valid range is : [0, GetMetadataCount()-1].

3.34.2.24 int HImage::GetSpecificMetadataCount (unsigned int *uID*)

Retrieve the number of metadata of a specific type attached to this image.

Returns

The metadata count.

Parameters

in	*uID*	The unique identifier of the metadata type to retrieve.

3.34.2.25 HMetadata* HImage::GetSpecificMetadata (unsigned int *uID*, int *index*)

Get one of the metadata of a specific type attached to the image.

Returns

The metadata.

Parameters

in	*uID*	The unique identifier of the metadata type to retrieve.
in	*index*	The index of the metadata,valid range is : [0, GetSpecificMetadataCount()-1].

3.34.2.26 void HImage::AddMetadata (const HMetadata & *metadata*)

Add metadata to the image.

Parameters

in	*metadata*	The metadata to add.

3.35 HImageColorProcessing Class Reference

This class applies a Bayer conversion on a raw image and/or does a color balance to an image.

```
#include <HImageColorProcessing.h>
```

Public Member Functions

- void EnableMonochromeConversion (bool enable)

  Enable monochrome conversion.
- bool IsMonochromeConversionEnabled ()

  Check if monochrome conversion is enabled.
- bool Get1DLut (int depth, bool color, unsigned short *red, unsigned short *green, unsigned short *blue)

  Get if 1D look up table value.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

COMMON FUNCTIONS

- bool ApplyColorProcessing (HImage *src, HImage *dest, bool *wasDoneInPlace)

  Apply the color processing on an image.
- bool ApplyColorProcessing (HImage *src, HImage *dest)

  Apply the color processing without altering the "src" image.

BAYER CONVERSION FUNCTIONS

- void EnableBayerConversion (bool enable)

  Enable or disable the Bayer conversion.
- bool IsBayerConversionEnabled ()

  Check if the module will apply Bayer conversion on source images.
- void SetPatternOrigin (eHBayerPatternOrigin value)

Set which Bayer pattern to use for the bayer conversion.

- eHBayerPatternOrigin GetPatternOrigin ()

  Get the current Bayer pattern used for the bayer conversion.
- void SetCalcMode (eHBayerCalculationMode value)

  Set the algorithm used to do the Bayer conversion. More precise calculations require more CPU time.
- eHBayerCalculationMode GetCalcMode ()

  Get the current algorithm used to do the Bayer conversion.
- void SetSampleFactor (eHBayerSampleFactor value)

  Set the sample factor applied during the Bayer conversion.
- eHBayerSampleFactor GetSampleFactor ()

  Get the current sample factor for the Bayer conversion.
- void EnableBGRxConversion (bool enable)

  Set the bayer conversion output format. BGR is the default format.
- bool IsBGRxConversionEnabled ()

  Check the bayer conversion output format.
- void EnableColorProcessingHQConversion (bool enable)

  Toggle the HQ conversion.
- bool IsColorProcessingHQConversionEnabled ()

  Check if HQ conversion is enabled.

MATRIX COLOR CORRECTION FUNCTIONS

Matrix color correction is applied as part of the bayer conversion.

- bool IsMatrixColorCorrectionEnabled ()

  Check if matrix color correction is enabled.
- void EnableMatrixColorCorrection (bool enable)

  Enable or disable matrix color correction.
- void GetCorrectionMatrix (double &factorBlueFromBlue, double &factorBlueFromGreen, double &factor↩BlueFromRed, double &factorBlue, double &factorGreenFromBlue, double &factorGreenFromGreen, double &factorGreenFromRed, double &factorGreen, double &factorRedFromBlue, double &factorRedFrom↩Green, double &factorRedFromRed, double &factorRed)

  Get the values of the matrix that is applied.
- void SetCorrectionMatrix (double factorBlueFromBlue, double factorBlueFromGreen, double factor↩BlueFromRed, double factorBlue, double factorGreenFromBlue, double factorGreenFromGreen, double factorGreenFromRed, double factorGreen, double factorRedFromBlue, double factorRedFromGreen, double factorRedFromRed, double factorRed)

  Set the values for the matrix that is applied.

COLOR BALANCE FUNCTIONS

- bool IsAutoColorBalanceEnabled ()

  Check if the color balance is automatic or manual.
- void EnableAutoColorBalance (bool enable)

  Set the type of color balance applied. The default is manual.
- void GetManualColorBalance (int &red, int &green, int &blue)

  Get the color balance offset values applied in manual mode. Values are in percent of the full image dynamic range.
- void SetManualColorBalance (int red, int green, int blue)

  Set the color balance offset values applied when in manual mode. Values are in percent of the full image dynamic range.
- eHAutoWhiteBalanceAlgorithm GetAutoColorBalanceAlgorithm ()

  Get the automatic color balance (white balance) algorithm applied.
- void SetAutoColorBalanceAlgorithm (eHAutoWhiteBalanceAlgorithm value)

  Set the automatic color balance (white balance) algorithm.
- void SetAutoWhiteBalanceROI (double offsetX, double offsetY, double width, double height)

  Set an ROI to calculate automatic color balance (white balance) parameters. Default is full image (0,0,100,100).
- void GetAutoWhiteBalanceROI (double &offsetX, double &offsetY, double &width, double &height)

  Get the ROI used to calculate the automatic color balance (white balance)

ASC CDL FUNCTIONS

American Society of Cinematographers Color Decision List (ASC CDL).

This define a format for the exchange of basic primary color grading information. A 3 parameters formula can be used to define most possible remapping and color correction:

```
* The formula for ASC CDL color correction is:
*
* where:
* - out is the color graded pixel code value
* - i is the input pixel code value (0=black, 1=white)
* - s is slope (any number 0 or greater, nominal value is 1.0)
* - o is offset (any number, nominal value is 0)
* - p is power (any number greater than 0, nominal value is 1.0)
*
* The formula is applied to the three color values for each pixel using the corresponding slope, offset, an
*
```

- void ResetASCSettings ()

  Reset ASC CDL settings to their default values.
- bool GetASCSettingsParam (double ascSlope[3], double ascOffset[3], double ascPower[3])

  Get current ASC parameters settings.
- bool SetASCSettingsParam (double ascSlope[3], double ascOffset[3], double ascPower[3])

  Set ASC Parameters.

LUT (Look-up table) PROCESSING FUNCTIONS

When possible the LUT processing will be applied at the same time as the Bayer conversion.

- bool IsLUTProcessEnabled ()

  Check if a LUT processing is applied on the images.
- void EnableLUTProcess (bool enable)

  Enable or disable the application of a LUT processing on the images.
- void RecalculateLUT ()

  Force the reinitialization of the LUT for LUT Processing.
- bool Is3DLookupTable ()

  Check if 3D LUT mode is enabled.
- void Enable3DLookupTable (bool d3dlut)

  Choose between a 3D LUT or a 1D LUT.
- bool SetCacheLUTFileName (LPCTSTR fileName, bool reload)

  Set filepath used to cache the current LUT data.
- bool AddThirdPartLutFile (LPCTSTR fileName)

  Insert a third-party LUT file to the LUT processing pipline.
- bool RemoveThirdPartLutFile (LPCTSTR fileName)

  Remove a third-party LUT file from LUT processing pipline.
- void SetColorClipLevel (int level)

  Set the color clip level.
- int GetColorClipLevel ()

  Get the current color clip level.
- void SetMonoPseudoColorMode (eMonoToPseudoColorLUT mode)

  Apply PseudoColor on monochrome images Force remapping of monochrome images to pseudo color, using some predefined LUT.
- eMonoToPseudoColorLUT GetMonoPseudoColorMode ()

  Get the current PseudoColor mode.
- bool Load1DPreDefinedLUT (int index)

  Load a predefined 1D LUT.
- bool LoadNpxTextLutFile (LPCTSTR fileName)

Load a comma separated LUT text file to define the remapping values.

- bool ExportLUTToTextFile (LPCTSTR fileName)

  Export the current LUT to a text file.
- static int Get1DSupportThirdPartLutFileCount ()

  Get the number of third-party 1D LUT file format.
- static bool Get1DSupportThirdPartLutFileFormatInfo (int index, LPTSTR desc, LPTSTR ext)

  Get third-party 1D LUT file format infomation.
- static int Get3DSupportThirdPartLutFileCount ()

  Get the number of third-party 3D LUT file format.
- static bool Get3DSupportThirdPartLutFileFormatInfo (int index, LPTSTR desc, LPTSTR ext)

  Get third-party 3D LUT file format infomation.
- static int Get1DPreDefinedLUTCount ()

  Get the number of predefined 1D LUT.
- static bool Get1DPreDefinedLUT (int index, LPTSTR name)

  Get information on each predefined 1D LUT.

3.35.1 Detailed Description

This class applies a Bayer conversion on a raw image and/or does a color balance to an image.

This class replaces HBayerConverter, now obsolete.

3.35.2 Member Function Documentation

3.35.2.1 void HImageColorProcessing::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.35.2.2 eHErrorCode HImageColorProcessing::GetLastError ( )

See also

HGrabModule::GetLastError

3.35.2.3 bool HImageColorProcessing::ApplyColorProcessing (HImage * *src,* HImage * *dest,* bool * *wasDoneInPlace*)

Apply the color processing on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*src*	The source image to be processed (must be a valid HImage).
in,out	*dest*	The processed image (must be a valid HImage, possibly created with HImage::CreateHImage()).
out	*wasDoneInPlace*	True if the color processing could be done directly on the source image (in that case the out is not used). False if "dest" had to be used (ex : Bayer conversion cannot be done in-place).

3.35.2.4 bool HImageColorProcessing::ApplyColorProcessing (HImage * *src*, HImage * *dest*)

Apply the color processing without altering the "src" image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*src*	The source image to be processed (must be a valid HImage).
in,out	*dest*	The processed image (must be a valid HImage, possibly created with HImage::CreateHImage()).

3.35.2.5 void HImageColorProcessing::EnableBayerConversion (bool *enable*)

Enable or disable the Bayer conversion.

Parameters

in	*enable*	True to enable the the Bayer conversion. False to disable it.

3.35.2.6 bool HImageColorProcessing::IsBayerConversionEnabled ( )

Check if the module will apply Bayer conversion on source images.

Returns

True if Bayer conversion is to be applied, false otherwise.

3.35.2.7 void HImageColorProcessing::SetPatternOrigin (eHBayerPatternOrigin *value*)

Set which Bayer pattern to use for the bayer conversion.

Parameters

in	*value*	The pattern to use for conversion.

3.35.2.8 eHBayerPatternOrigin HImageColorProcessing::GetPatternOrigin ( )

Get the current Bayer pattern used for the bayer conversion.

Returns

The current Bayer pattern.

3.35.2.9 void HImageColorProcessing::SetCalcMode (eHBayerCalculationMode *value*)

Set the algorithm used to do the Bayer conversion. More precise calculations require more CPU time.

Parameters

in	*value*	The algorithm to use.

3.35.2.10 eHBayerCalculationMode HImageColorProcessing::GetCalcMode ( )

Get the current algorithm used to do the Bayer conversion.

Returns

The current algorithm.

3.35.2.11 void HImageColorProcessing::SetSampleFactor (eHBayerSampleFactor *value*)

Set the sample factor applied during the Bayer conversion.

Default is 1:1. This allows to do the bayer conversion faster at the expense of a smaller output image. The sample factor is applied only if the bayer algorithm is H_BAYER_CALCMODE_FAST.

Parameters

in	*value*	The sample factor value.

3.35.2.12 eHBayerSampleFactor HImageColorProcessing::GetSampleFactor ( )

Get the current sample factor for the Bayer conversion.

Returns

The current sample factor.

3.35.2.13 void HImageColorProcessing::EnableBGRxConversion (bool *enable*)

Set the bayer conversion output format. BGR is the default format.

Parameters

in	*enable*	True for BGRx, false for BGR.

3.35.2.14 bool HImageColorProcessing::IsBGRxConversionEnabled ( )

Check the bayer conversion output format.

Returns

True if the Bayer conversion output a BGRx format instead of BGR, false otherwise.

3.35.2.15 void HImageColorProcessing::EnableColorProcessingHQConversion (bool *enable*)

Toggle the HQ conversion.

By default, bayer conversion will output in BGR24. However, if the source image bit depth is greater than 8bpp and HQ conversion is enabled, the output image will be BGR48. This setting is ignored if the Bayer conversion outputs BGRx.

See also

EnableBayerConversionToBGRx

Parameters

in	*enable*	True to convert to BGR48 as needed, false to go BGR24.

3.35.2.16 bool HImageColorProcessing::IsColorProcessingHQConversionEnabled ( )

Check if HQ conversion is enabled.

Returns

True if the Bayer conversion convert >8 bits images to 48 bits BGR.

3.35.2.17 bool HImageColorProcessing::IsMatrixColorCorrectionEnabled ( )

Check if matrix color correction is enabled.

Returns

True if color matrix correction is enabled.

3.35.2.18 void HImageColorProcessing::EnableMatrixColorCorrection (bool *enable*)

Enable or disable matrix color correction.

Parameters

in	*enable*	True to enable the color Matrix correction.

3.35.2.19 void HImageColorProcessing::GetCorrectionMatrix (double & *factorBlueFromBlue,* double & *factorBlueFromGreen,* double & *factorBlueFromRed,* double & *factorBlue,* double & *factorGreenFromBlue,* double & *factorGreenFromGreen,* double & *factorGreenFromRed,* double & *factorGreen,* double & *factorRedFromBlue,* double & *factorRedFromGreen,* double & *factorRedFromRed,* double & *factorRed*)

Get the values of the matrix that is applied.

Parameters

out	*factorBlue↩ FromBlue*	'Blue from Blue' Factor
out	*factorBlue↩ FromGreen*	'Blue from Green' Factor
out	*factorBlue↩ FromRed*	'Blue from Red' Factor
out	*factorBlue*	'Blue' Factor
out	*factorGreen↩ FromBlue*	'Green from Blue' Factor
out	*factorGreen↩ FromGreen*	'Green from Green' Factor
out	*factorGreen↩ FromRed*	'Green from Red' Factor
out	*factorGreen*	'Green' Factor
out	*factorRed↩ FromBlue*	'Red from Blue' Factor
out	*factorRed↩ FromGreen*	'Red from Green' Factor
out	*factorRed↩ FromRed*	'Red from Red' Factor
out	*factorRed*	'Red' Factor

3.35.2.20 void HImageColorProcessing::SetCorrectionMatrix (double *factorBlueFromBlue,* double *factorBlueFromGreen,* double *factorBlueFromRed,* double *factorBlue,* double *factorGreenFromBlue,* double *factorGreenFromGreen,* double *factorGreenFromRed,* double *factorGreen,* double *factorRedFromBlue,* double *factorRedFromGreen,* double *factorRedFromRed,* double *factorRed*)

Set the values for the matrix that is applied.

Parameters

in	*factorBlue↩ FromBlue*	'Blue from Blue' Factor
in	*factorBlue↩ FromGreen*	'Blue from Green' Factor
in	*factorBlue↩ FromRed*	'Blue from Red' Factor
in	*factorBlue*	'Blue' Factor
in	*factorGreen↩ FromBlue*	'Green from Blue' Factor
in	*factorGreen↩ FromGreen*	'Green from Green' Factor
in	*factorGreen↩ FromRed*	'Green from Red' Factor
in	*factorGreen*	'Green' Factor
in	*factorRed↩ FromBlue*	'Red from Blue' Factor
in	*factorRed↩ FromGreen*	'Red from Green' Factor
in	*factorRed↩ FromRed*	'Red from Red' Factor

in	*factorRed*	'Red' Factor

3.35.2.21 bool HImageColorProcessing::IsAutoColorBalanceEnabled ( )

Check if the color balance is automatic or manual.

Returns

True if the color balance is automatic. False if manual.

3.35.2.22 void HImageColorProcessing::EnableAutoColorBalance (bool *enable*)

Set the type of color balance applied. The default is manual.

Parameters

in	*enable*	True to apply automatic color balance, false for manual color balance.

3.35.2.23 void HImageColorProcessing::GetManualColorBalance (int & *red,* int & *green,* int & *blue*)

Get the color balance offset values applied in manual mode. Values are in percent of the full image dynamic range.

Parameters

out	*red*	Offset value applied to the red component.
out	*green*	Offset value applied to the green component.
out	*blue*	Offset value applied to the blue component.

3.35.2.24 void HImageColorProcessing::SetManualColorBalance (int *red,* int *green,* int *blue*)

Set the color balance offset values applied when in manual mode. Values are in percent of the full image dynamic range.

Parameters

in	*red*	Offset value applied to the red component.
in	*green*	Offset value applied to the green component.
in	*blue*	Offset value applied to the blue component.

3.35.2.25 eHAutoWhiteBalanceAlgorithm HImageColorProcessing::GetAutoColorBalanceAlgorithm ( )

Get the automatic color balance (white balance) algorithm applied.

Returns

The color balance algorithm.

3.35.2.26 void HImageColorProcessing::SetAutoColorBalanceAlgorithm (eHAutoWhiteBalanceAlgorithm *value*)

Set the automatic color balance (white balance) algorithm.

Parameters

in	*value*	The algorithm to use.

3.35.2.27 void HImageColorProcessing::SetAutoWhiteBalanceROI (double *offsetX,* double *offsetY,* double *width,* double *height*)

Set an ROI to calculate automatic color balance (white balance) parameters. Default is full image (0,0,100,100).

Parameters

in	*offsetX*	The horizontal offset percentage of the region.
in	*offsetY*	The vertical offset percentage of the region.
in	*width*	The width percentage of the region.
in	*height*	The height percentage of the region.

3.35.2.28 void HImageColorProcessing::GetAutoWhiteBalanceROI (double & *offsetX,* double & *offsetY,* double & *width,* double & *height*)

Get the ROI used to calculate the automatic color balance (white balance)

Parameters

out	*offsetX*	The horizontal offset percentage of the region.
out	*offsetY*	The vertical offset percentage of the region.
out	*width*	The width percentage of the region.
out	*height*	The height percentage of the region.

3.35.2.29 bool HImageColorProcessing::GetASCSettingsParam (double *ascSlope[3],* double *ascOffset[3],* double *ascPower[3]*)

Get current ASC parameters settings.

Array index 0 is the blue channel; index 1 is the green channel and index 3 is the red channel. Only use index 0 when When processing monochrome images.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*ascSlope*	ASC slope parameter.
out	*ascOffset*	ASC offset parameter.
out	*ascPower*	ASC power parameter.

3.35.2.30 bool HImageColorProcessing::SetASCSettingsParam (double *ascSlope[3],* double *ascOffset[3],* double *ascPower[3]*)

Set ASC Parameters.

Array index 0 is the blue channel; index 1 is the green channel and index 3 is the red channel. Only use index 0 when When processing monochrome images.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*ascSlope*	ASC slope parameter.
in	*ascOffset*	ASC offset parameter.
in	*ascPower*	ASC power parameter.

3.35.2.31 bool HImageColorProcessing::IsLUTProcessEnabled ( )

Check if a LUT processing is applied on the images.

Returns

True if LUT processing is enabled, false otherwise.

3.35.2.32 void HImageColorProcessing::EnableLUTProcess (bool *enable*)

Enable or disable the application of a LUT processing on the images.

Parameters

in	*enable*	True to apply the LUT processing processing.

3.35.2.33 bool HImageColorProcessing::Is3DLookupTable ( )

Check if 3D LUT mode is enabled.

Returns

True if 3D LUT mode enabled else 1D LUT enabled.

3.35.2.34 void HImageColorProcessing::Enable3DLookupTable (bool *d3dlut*)

Choose between a 3D LUT or a 1D LUT.

Parameters

in	*d3dlut*	True to use a 3D LUT, false to use a 1D LUT.

3.35.2.35 bool HImageColorProcessing::SetCacheLUTFileName (LPCTSTR *fileName,* bool *reload*)

Set filepath used to cache the current LUT data.

LUT processing can store current LUT data to a file for future use to avoid unnecessary recalculations.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*fileName*	Filepath to the LUT cache file. NULL to disable this feature.
in	*reload*	True to immediately reload the data.

3.35.2.36 static int HImageColorProcessing::Get1DSupportThirdPartLutFileCount ( ) [static]

Get the number of third-party 1D LUT file format.

Returns

The number of 1D third-party LUT file format.

3.35.2.37 static bool HImageColorProcessing::Get1DSupportThirdPartLutFileFormatInfo (int *index*, LPTSTR *desc*, LPTSTR *ext*) [static]

Get third-party 1D LUT file format infomation.

Returns

True if the function is successful, false otherwise.

Parameters

in	*index*	The index of file format to query.
out	*desc*	The description of the LUT file format.
out	*ext*	The extension of the file.

3.35.2.38 static int HImageColorProcessing::Get3DSupportThirdPartLutFileCount ( ) [static]

Get the number of third-party 3D LUT file format.

Returns

The number of third-party 3D LUT file format.

3.35.2.39 static bool HImageColorProcessing::Get3DSupportThirdPartLutFileFormatInfo (int *index*, LPTSTR *desc*, LPTSTR *ext*) [static]

Get third-party 3D LUT file format infomation.

Returns

True if the function is successful, false otherwise.

Parameters

in	*index*	The index of file format to query.
out	*desc*	The description for for LUT file format.
out	*ext*	The extension of the file.

3.35.2.40 bool HImageColorProcessing::AddThirdPartLutFile (LPCTSTR *fileName*)

Insert a third-party LUT file to the LUT processing pipline.

Returns

True if the function is successful, false otherwise.

Parameters

in	*fileName*	Filepath to the LUT file.

3.35.2.41 bool HImageColorProcessing::RemoveThirdPartLutFile (LPCTSTR *fileName*)

Remove a third-party LUT file from LUT processing pipline.

Returns

True if the function is successful, false otherwise.

Parameters

in	*fileName*	Filepath to the LUT file.

3.35.2.42 void HImageColorProcessing::SetColorClipLevel (int *level*)

Set the color clip level.

This function is only available for bit depth greater than 8 bits. This function can highlight specific bit ranges on the images. By default, HImageConvert always return the 8 most significant bits of captured images.

```
* As an example, a 10 bit image can be leveled in 3 ways:
* - level 2: 8 most significant bits (bit 2 to 10, default HImageConvert setting)
* - level 1: 8 middle significant bits (bit 1 to 9)
* - level 0: 8 least significant bits (bit 0 to 7)
*
```

Parameters

in	*level*	The clip level value. Use -1, to disable this feature and return to default mode.

3.35.2.43 int HImageColorProcessing::GetColorClipLevel ( )

Get the current color clip level.

Returns

The color clip level.

3.35.2.44 void HImageColorProcessing::SetMonoPseudoColorMode (eMonoToPseudoColorLUT *mode*)

Apply PseudoColor on monochrome images Force remapping of monochrome images to pseudo color, using some predefined LUT.

If the source image is color, if bayer conversion is enabled or if MonochromeConversionEnabled is enabled, this function has no effect.

```
* It can be set to:
* - H_MONO_TO_PSEUDO_COLOR_DISABLED : disabled.
* - H_MONO_TO_PSEUDO_COLOR_LUT_RAINBOW : rainbow.
* - H_MONO_TO_PSEUDO_COLOR_LUT_INVERTED_RAINBOW : inverted rainbow.
* - H_MONO_TO_PSEUDO_COLOR_LUT_HOT : Hot color.
* - H_MONO_TO_PSEUDO_COLOR_LUT_COLD : cold color.
* - H_MONO_TO_PSEUDO_COLOR_LUT_NEGATIVE : negative color.
*
```

Parameters

in	*mode*	Pseudo mode.

3.35.2.45 eMonoToPseudoColorLUT HImageColorProcessing::GetMonoPseudoColorMode ( )

Get the current PseudoColor mode.

Returns

The current mode.

3.35.2.46 static int HImageColorProcessing::Get1DPreDefinedLUTCount ( ) [static]

Get the number of predefined 1D LUT.

Returns

The numbers of predefined 1D LUT.

3.35.2.47 static bool HImageColorProcessing::Get1DPreDefinedLUT (int *index*, LPTSTR *name*) [static]

Get information on each predefined 1D LUT.

Returns

True if the function is successful, false otherwise.

Parameters

in	*index*	The index of predefined LUT to query.
out	*name*	The name of predefined LUT.

3.35.2.48 bool HImageColorProcessing::Load1DPreDefinedLUT (int *index*)

Load a predefined 1D LUT.

Returns

True if the function is successful, false otherwise.

Parameters

in	*index*	The index of the file format to load.

3.35.2.49 bool HImageColorProcessing::LoadNpxTextLutFile (LPCTSTR *fileName*)

Load a comma separated LUT text file to define the remapping values.

The text file syntax is very simple. Each contains 4 entries:

- Level value to be remapped.
- Blue value.
- Green value.
- Red values. There is no need to specify all the LUT values and entries. Only the needed values can be specified. All missing value will not be remapped. Level can be specified as range using [-] characters, comment a line, Out bound values are ignored. Example:

```
 * Index, Blue, Green, Red Remapping value
* 0, 255, 0, 0  : gray level 0 will be remapping as pure blue,
* 1, 10, 10,  , : gray level 1 will be remapping as Blue = 10, Green = 10, Red default (1),
* [200,255], 255, 255, 255: gray levels 200 to 255 are remap to pure white
*
```

Returns

True if the function is successful or false otherwise.

Parameters

in	*fileName*	The LUT filepath.

3.35.2.50 bool HImageColorProcessing::ExportLUTToTextFile (LPCTSTR *fileName*)

Export the current LUT to a text file.

See also

LoadNpxTextLutFile

Returns

True if the function is successful or false if the function failed.

Parameters

in	*fileName*	The LUT filepath.

3.35.2.51 void HImageColorProcessing::EnableMonochromeConversion (bool *enable*)

Enable monochrome conversion.

If the source image is monochrome and Bayer conversion is disabled, enabling monochrome conversion will force the output image to be a monochrome image.

Parameters

in	*enable*	True for force to monochrome

3.35.2.52 bool HImageColorProcessing::IsMonochromeConversionEnabled ( )

Check if monochrome conversion is enabled.

Returns

True if the monochrone conversion is enabled.

3.35.2.53 bool HImageColorProcessing::Get1DLut (int *depth*, bool *color*, unsigned short * *red*, unsigned short * *green*, unsigned short * *blue*)

Get if 1D look up table value.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*depth*	The depth of 1D look up table
in	*color*	True if want color look up table
out	*red*	values of red channels
out	*green*	values of green channels
out	*blue*	values of blue channels

3.36 HImageConverter Class Reference

This class converts an HImage from one format to another.

```
#include <HImageConverter.h>
```

Public Member Functions

- bool ConvertToBGR (HImage *imageIn, HImage *imageOut, bool flip=false)

  Converts an image to the BGR format with 24 bits per pixel (Blue-Green-Red).
- bool ConvertToBGRx (HImage *imageIn, HImage *imageOut)

  Converts an image to the BGRx format with 32 bits per pixel (Blue-Green-Red-Filler).
- bool ConvertToMono8 (HImage *imageIn, HImage *imageOut, bool flip=false)

  Convert an image to the Mono 8 format.
- void ValidateAllocation (bool enable)

  Set the buffer allocation validation, forcing a buffer overflow check and reallocating memory as needed.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.36.1 Detailed Description

This class converts an HImage from one format to another.

This class can be used to convert any supported image format to the following image formats : Monochrome 8 bits (most basic monochrome image), Color BGR (most basic color image) and Color BGRx (color format compatible with most Win32 display/bitmap functions).

This can be useful for image processing by allowing the use of a single algorithm for all converted images instead of using different ones for each image format. For example, HImageExporter and HAVIExporter internally uses those functions to insure compatibility of the images with the exporting libraries.

Unless specified, any source image that is more than 8 bit depth will be converted to an 8 bit or 24 bit color image. The 8 most significant bit are kept.

Note that raw bayer images when converted to BRG will becomes raw bayer RGB images. (No Bayer interpolation will be performed.)

Warning : All color remapping functions are now obsolete please use HImageProcessing instead.

3.36.2 Member Function Documentation

3.36.2.1 void HImageConverter::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.36.2.2 eHErrorCode HImageConverter::GetLastError ( )

See also

HGrabModule::GetLastError

3.36.2.3 bool HImageConverter::ConvertToBGR (HImage * *imageIn*, HImage * *imageOut*, bool *flip* = `false`)

Converts an image to the BGR format with 24 bits per pixel (Blue-Green-Red).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*imageIn*	The source image.
out	*imageOut*	Where the converted image is stored. The related target is created with HImage::CreateHImage().
in	*flip*	If this is true, the image will be also be horizontally flipped. It is required for some image libraries but some input formats are not supported yet .

3.36.2.4 bool HImageConverter::ConvertToBGRx (HImage * *imageIn*, HImage * *imageOut*)

Converts an image to the BGRx format with 32 bits per pixel (Blue-Green-Red-Filler).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*imageIn*	The source image.
out	*imageOut*	Where the converted image is stored. The related target is created with HImage::CreateHImage().

3.36.2.5 bool HImageConverter::ConvertToMono8 (HImage * *imageIn*, HImage * *imageOut*, bool *flip* = `false`)

Convert an image to the Mono 8 format.

This function will only accept Monochrome images input.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*imageIn*	The source image.
out	*imageOut*	Where the converted image is stored. The related target is created with HImage::CreateHImage().
in	*flip*	If this is true, the image will be also be horizontally flipped (upside down). Some image libraries require this.

3.37 HImageExporter Class Reference

This class takes one or more images in HImage format and saves them in stand-alone files on disk.

```
#include <HImageExporter.h>
```

Public Member Functions

- bool AddJPEGMetadata (unsigned short tag, unsigned short type, unsigned long count, const void *data)

  Add EXIF data to exported JPEG images.
- void EnableTIFFMetadata (bool enable)

  Enable or disable EXIF export for TIFF.
- eHImageDateTimeFormat GetDateTimeFormat ()

  Get the current Date/Time format used for TIFF DATETIME tag.
- void SetDateTimeFormat (eHImageDateTimeFormat format)

  Set the Date/Time format used for TIFF DATETIME tag.
- bool SetSeparator (TCHAR separator)

  Change the default separator used between the text string and the index number.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

EXPORT FUNCTIONS.

This function set will generate the files names based on the name of the first file.

- bool BeginExportBmp (LPCTSTR filename, int digits=1, unsigned int startIndex=1)

Prepare the module to successively save images in BMP format.

- ∗bool BeginExportJpeg (LPCTSTR filename, int digits=1, unsigned int startIndex=1)

  Prepare the module to successively save images in JPEG format.
- bool BeginExportTiff (LPCTSTR filename, bool multiTIFF=false, int digits=1, unsigned int startIndex=1)

  Prepare the module to successively save images in TIFF format.
- bool BeginExportPng (LPCTSTR filename, int digits=1, unsigned int startIndex=1)

  Prepare the module to successively save images in PNG format.
- bool BeginExportJpeg2k (LPCTSTR filename, int digits=1, unsigned int startIndex=1)

  Prepare the module to successively save images in JPEG2000 format.
- bool BeginExportFit (LPCTSTR filename, bool multiFit=false, bool exportColor=false, int digits=1, unsigned int startIndex=1)

  Prepare the module to successively save images in Fit format.
- bool BeginExportDpx (LPCTSTR filename, bool force10bitBGR=false, int digits=1, unsigned int start↩Index=1)

  Prepare the module to successively save images in Dpx format.
- bool BeginExportDng (LPCTSTR filename, eHBayerPatternOrigin pattern=H_BAYER_PATTERNORIG↩IN_GB, double frameRate=0, LPCTSTR camProfile=NULL, int digits=1, unsigned int startIndex=1)

  Prepare the module to successively save images in Dng format.
- bool BeginExportJpegXR (LPCTSTR filename, int quality, int digits=1, unsigned int startIndex=1)

  Prepare the module to successively save images in JPEG XR format.
- bool ExportImage (HImage ∗image, LPTSTR exportedFileName=NULL)

  ExportImage exports an image to a preselected format.
- bool EndExport ()

  EndExport completes the export procedure and executes to some clean up.

'EXPORT TO' FUNCTIONS.

This function set allows the user to individually name each and every exported file.

- bool BeginExportBmpTo ()

  Prepare the module to save images successively in the BMP format.
- bool BeginExportJpegTo ()

  Prepare the module to successively save images in JPEG format.
- bool BeginExportTiffTo ()

  Prepare the module to successively save images in TIFF format.
- bool BeginExportPngTo ()

  Prepare the module to successively save images in PNG format.
- bool BeginExportJpeg2kTo ()

  Prepare the module to successively save images in Jpeg2000 format.
- bool BeginExportFitTo (bool exportColor=false)

  Prepare the module to successively save images in Fit format.
- bool BeginExportDpxTo (bool force10bitBGR=false)

  Prepare the module to successively save images in Dpx format.
- bool BeginExportDngTo (eHBayerPatternOrigin pattern=H_BAYER_PATTERNORIGIN_GB, double frameRate=0, LPCTSTR camProfile=NULL)

  Prepare the module to successively save images in Dng format.
- bool BeginExportJpegXRTo (int quality)

  Prepare the module to successively save images in JPEG XR format.
- bool ExportImageTo (HImage ∗image, LPCTSTR fileName)

  ExportImageTo exports an image to a preselected format.
- bool EndExportTo ()

  EndExportTo completes the export procedure and executes some cleanup.

SETUP FUNCTIONS

- bool SetJPEGQuality (eHJpegQuality quality)

  Set the JPEG compression quality used for images exported to JPEG format.
- eHJpegQuality GetJPEGQuality ()

  Get the current JPEG compression quality.

GPU JPEG FUNCTIONS

- void SetJpegEncoderType (eJpegImageExportType type)

  Set Jpeg encoder type encoder type can be IPP(CPU),and GPU.
- bool SetJpegGPUEncoderType (eDeviceAcclType type, int deviceIndex=-1)

  Set Video Sequence Ecoder type, current support CPU,Intel and Nvidia or AMD.
- void SetJpegGPUCompressionBayerPatternType (eHBayerPatternOrigin pattern)

  Set the bayer pattern for the GPU lossy JPEG and Cayer encoders.
- void EnableGPUJpegCompressionBayerInterp (bool enable)

  Set enable or disable Bayer interpolation for RAW bayer image CUDA Jpeg compresssion.
- void EnableGPUJpegCompressionColorprcessingLUT (bool enable)

  Set enable or disable color procession for CUDA Jpeg compresssion True if enable or false if disable.
- bool UpdateGPUJpegCompressionLUT (HImageColorProcessing *processor)

  Update color procession lookup table for CUDA Jpeg compresssion.

3.37.1 Detailed Description

This class takes one or more images in HImage format and saves them in stand-alone files on disk.

3.37.2 Member Function Documentation

3.37.2.1 void HImageExporter::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.37.2.2 eHErrorCode HImageExporter::GetLastError ( )

See also

HGrabModule::GetLastError

3.37.2.3 bool HImageExporter::BeginExportBmp (LPCTSTR *filename*, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in BMP format.

Use in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.bmp")
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.bmp, filename002.bmp, filename003.bmp, filename004.bmp, etc.
in	*startIndex*	The start index.

3.37.2.4 ∗ bool HImageExporter::BeginExportJpeg (LPCTSTR *filename*, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in JPEG format.

Use in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.jpg")
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.jpg, filename002.jpg, filename003.jpg, filename004.jpg, etc.
in	*startIndex*	The start index.

3.37.2.5 bool HImageExporter::BeginExportTiff (LPCTSTR *filename*, bool *multiTIFF* = false, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in TIFF format.

Use in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.tif")
in	*multiTIFF*	If true, successive images will be saved in a single multipaged TIFF file. If false, each image will be saved in its own TIFF file.
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.tif, filename002.tif, filename003.tif, filename004.tif, etc.
in	*startIndex*	The start index.

3.37.2.6 bool HImageExporter::BeginExportPng (LPCTSTR *filename*, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in PNG format.

Use in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.png")
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.png, filename002.png, filename003.png, filename004.png, etc.
in	*startIndex*	The start index.

3.37.2.7 bool HImageExporter::BeginExportJpeg2k (LPCTSTR *filename*, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in JPEG2000 format.

To be used in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.jp2")
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.jp2, filename002.jp2, filename003.jp2, filename004.jp2, etc.
in	*startIndex*	The start index.

3.37.2.8 bool HImageExporter::BeginExportFit (LPCTSTR *filename*, bool *multiFit* = false, bool *exportColor* = false, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in Fit format.

Use in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.fit")
in	*multiFit*	If true, successive images will be saved in a single multipaged Fit file. If false, each image will be saved in its own Fit file.
in	*exportColor*	If true, successive images will be divided into 3 individual data files: red, green and blue. The color plan is appended to the file name.
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.fit, filename002.fit, filename003.fit, filename004.fit, etc.
in	*startIndex*	The start index.

3.37.2.9 bool HImageExporter::BeginExportDpx (LPCTSTR *filename*, bool *force10bitBGR* = false, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in Dpx format.

Use in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.dpx")
in	*force10bitBGR*	True to force output as 10 bit BGR format
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.dpx, filename002.dpx, filename003.dpx, filename004.dpx, etc.
in	*startIndex*	The start index.

3.37.2.10 bool HImageExporter::BeginExportDng (LPCTSTR *filename*, eHBayerPatternOrigin *pattern* = H_BAYER_PATTERNORIGIN_GB, double *frameRate* = 0, LPCTSTR *camProfile* = NULL, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in Dng format.

Dng only support RAW format Use in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.dng")
in	*pattern*	Bayer pattern for raw image
in	*frameRate*	camera frame rate
in	*camProfile*	camera profile
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.dng, filename002.dng, filename003.dng, filename004.dng, etc.
in	*startIndex*	The start index.

3.37.2.11 bool HImageExporter::BeginExportJpegXR (LPCTSTR *filename*, int *quality*, int *digits* = 1, unsigned int *startIndex* = 1)

Prepare the module to successively save images in JPEG XR format.

Use in conjunction with ExportImage() and EndExport().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*filename*	The full path name of the destination file. (Ex: "C:\folder\image.wdp")
in	*quality*	: Image quality
in	*digits*	The number of digits to append at the end of the filename. For example, if digits is 3, files will be exported as : filename001.wdp, filename002.wdp, filename003.wdp, filename004.wdp, etc.
in	*startIndex*	The start index.

3.37.2.12 bool HImageExporter::ExportImage (HImage * *image*, LPTSTR *exportedFileName* = NULL)

ExportImage exports an image to a preselected format.

The format is selected by calling a "BeginExport..." function.

Example :

```
HImageExporter exporter;
char filename[MAX_PATH];
exporter.BeginExportBMP("C:\\image.bmp", 1);
exporter.ExportImage(image1); //creates image1.bmp
exporter.ExportImage(image2); //creates image2.bmp
exporter.ExportImage(image3); //creates image3.bmp
exporter.EndExport();
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The image to be exported (must be a valid HImage).
out	*exportedFile↩ Name*	The name of the image file written on disk. (if NULL, name is not returned)

3.37.2.13 bool HImageExporter::EndExport ( )

EndExport completes the export procedure and executes to some clean up.

Use in conjunction with the BeginExportXXX functions.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.14 bool HImageExporter::BeginExportBmpTo ( )

Prepare the module to save images successively in the BMP format.

Use in conjunction with ExportImageTo() and EndExportTo().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.15 bool HImageExporter::BeginExportJpegTo ( )

Prepare the module to successively save images in JPEG format.

Use in conjunction with ExportImageTo() and EndExportTo().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.16 bool HImageExporter::BeginExportTiffTo ( )

Prepare the module to successively save images in TIFF format.

Use in conjunction with ExportImageTo() and EndExportTo(). For multi-paged tiff files use BeginExportTiff().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.17 bool HImageExporter::BeginExportPngTo ( )

Prepare the module to successively save images in PNG format.

Use in conjunction with ExportImageTo() and EndExportTo().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.18 bool HImageExporter::BeginExportJpeg2kTo ( )

Prepare the module to successively save images in Jpeg2000 format.

Use in conjunction with ExportImageTo() and EndExportTo().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.19 bool HImageExporter::BeginExportFitTo (bool *exportColor* = `false`)

Prepare the module to successively save images in Fit format.

Use in conjunction with ExportImageTo() and EndExportTo().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

`in`	*exportColor*	If true, successive images will be divided into 3 individual data files: red, green and blue. The color plan is appended to the file name.

3.37.2.20 bool HImageExporter::BeginExportDpxTo (bool *force10bitBGR* = `false`)

Prepare the module to successively save images in Dpx format.

Use in conjunction with ExportImageTo() and EndExportTo().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.21 bool HImageExporter::BeginExportDngTo (eHBayerPatternOrigin *pattern* = H_BAYER_PATTERNORIGIN_GB, double *frameRate* = 0, LPCTSTR *camProfile* = NULL)

Prepare the module to successively save images in Dng format.

Use in conjunction with ExportImageTo() and EndExportTo().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pattern*	Bayer pattern for raw image
in	*frameRate*	camera frame rate
in	*camProfile*	camera profile

3.37.2.22 bool HImageExporter::BeginExportJpegXRTo (int *quality*)

Prepare the module to successively save images in JPEG XR format.

Use in conjunction with ExportImageTo() and EndExportTo().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*quality*	: Image quality

3.37.2.23 bool HImageExporter::ExportImageTo (HImage * *image*, LPCTSTR *fileName*)

ExportImageTo exports an image to a preselected format.

The format is selected by calling a "BeginExport...To" function.

Example :

```
HImageExporter exporter;
char filename[MAX_PATH];
exporter.BeginExportBmpTo();
exporter.ExportImageTo(image, "C:\\image.bmp");
exporter.ExportImageTo(image2, "C:\\other-image.bmp");
exporter.EndExportTo();
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The image to be exported (must be a valid HImage).
in	*fileName*	The name of the image file to create on disk.

3.37.2.24 bool HImageExporter::EndExportTo ( )

EndExportTo completes the export procedure and executes some cleanup.

Use in conjunction with the BeginExportXXXTo functions.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.25 bool HImageExporter::SetJPEGQuality (eHJpegQuality *quality*)

Set the JPEG compression quality used for images exported to JPEG format.

It will not affect any format other than JPEG. Will regulate the resulting file size, image quality and compression speed. Experimenting with the different compression schemes will allow to find the most appropriate method for the user's needs.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*quality*	The JPEG quality.

3.37.2.26 eHJpegQuality HImageExporter::GetJPEGQuality ( )

Get the current JPEG compression quality.

Returns

The JPEG compression quality.

3.37.2.27 void HImageExporter::SetJpegEncoderType (eJpegImageExportType *type*)

Set Jpeg encoder type encoder type can be IPP(CPU),and GPU.

Parameters

in	*type*	Jpeg encoder type

3.37.2.28 bool HImageExporter::SetJpegGPUEncoderType (eDeviceAcclType *type*, int *deviceIndex* = −1)

Set Video Sequence Ecoder type, current support CPU,Intel and Nvidia or AMD.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*type*	The H264 encoder type
in	*deviceIndex*	device index for Nvidia device

3.37.2.29 void HImageExporter::SetJpegGPUCompressionBayerPatternType (eHBayerPatternOrigin *pattern*)

Set the bayer pattern for the GPU lossy JPEG and Cayer encoders.

GPU lossy JPEG and Cayer encoder support Bayer interpolation for RAW bayer image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pattern*	bayer pattern

3.37.2.30 void HImageExporter::EnableGPUJpegCompressionBayerInterp (bool *enable*)

Set enable or disable Bayer interpolation for RAW bayer image CUDA Jpeg compresssion.

True if enable or false if disable

Parameters

in	*enable*	mode for GPU bayer interpolation

3.37.2.31 void HImageExporter::EnableGPUJpegCompressionColorprcessingLUT (bool *enable*)

Set enable or disable color procession for CUDA Jpeg compresssion True if enable or false if disable.

Parameters

in	*enable*	mode for GPU color procession

3.37.2.32 bool HImageExporter::UpdateGPUJpegCompressionLUT (HImageColorProcessing * *processor*)

Update color procession lookup table for CUDA Jpeg compresssion.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.37.2.33 bool HImageExporter::AddJPEGMetadata (unsigned short *tag,* unsigned short *type,* unsigned long *count,* const void * *data*)

Add EXIF data to exported JPEG images.

EXIF support (when using Intel IPP to export to JPEG only).

See also

SetJPEGLib()

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*tag*	EXIF tag ID
in	*type*	EXIF tag value type
in	*count*	size of data
in	*data*	EXIF data buffer

3.37.2.34 void HImageExporter::EnableTIFFMetadata (bool *enable*)

Enable or disable EXIF export for TIFF.

Parameters

in	*enable*	if true, EXIF will be exported for TIFF files

3.37.2.35 eHImageDateTimeFormat HImageExporter::GetDateTimeFormat ( )

Get the current Date/Time format used for TIFF DATETIME tag.

Returns

The DATETIME format.

3.37.2.36 void HImageExporter::SetDateTimeFormat (eHImageDateTimeFormat *format*)

Set the Date/Time format used for TIFF DATETIME tag.

Parameters

in	*format*	The DATETIME format to use

3.37.2.37 bool HImageExporter::SetSeparator (TCHAR *separator*)

Change the default separator used between the text string and the index number.

Can't be one of those : "/\:∗?<>|". Default is "-".

Parameters

in	*separator*	The separator character to use.

3.38 HImageMerge Class Reference

Merge 2 images together to create a larger image.

```
#include <HImageMerge.h>
```

Public Member Functions

- void SetMergeMode (eHMerge_Mode mode)

  Set the merge mode.
- eHMerge_Mode GetMergeMode ()

  Get the current merge mode.
- void ForceOutputBRG24 (bool bgr)

  Force image merge output as BGR24 format.
- bool CheckImageMergeFormat (eHImageFormat format1, int bitDepth1, eHImageFormat format2, int bit↩ Depth2)

  Check if the image format and bit depth are supported by the merge function.
- bool Merge (HImage ∗srcImage, HImage ∗srcImage2, HImage ∗destImage)

  Merge two images.
- bool MultipleMerge (HImage ∗∗imageList, int size, int col, int row, HImage ∗destImage)

  Merge multiple images.
- bool Stitch (HImage ∗srcImage, int offsetSrcX, int offsetSrcY, int sizeSrcX, int sizeSrcY, HImage ∗destImage, int offsetDestX, int offsetDestY)

  Stitch source image to target image.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.38.1 Detailed Description

Merge 2 images together to create a larger image.

3.38.2 Member Function Documentation

3.38.2.1 void HImageMerge::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.38.2.2 eHErrorCode HImageMerge::GetLastError ( )

See also

HGrabModule::GetLastError

3.38.2.3 void HImageMerge::SetMergeMode (eHMerge_Mode *mode*)

Set the merge mode.

Parameters

in	*mode*	The merge mode.

3.38.2.4 eHMerge_Mode HImageMerge::GetMergeMode ( )

Get the current merge mode.

Returns

The merge mode.

3.38.2.5 void HImageMerge::ForceOutputBRG24 (bool *bgr*)

Force image merge output as BGR24 format.

Parameters

in	*bgr*	True to force output to BGR24 format. False to keep the source format.

3.38.2.6 bool HImageMerge::CheckImageMergeFormat (eHImageFormat *format1*, int *bitDepth1*, eHImageFormat *format2*, int *bitDepth2*)

Check if the image format and bit depth are supported by the merge function.

Returns

True if the image format and bit depth combination is supported, false otherwise.

Parameters

in	*format1*	First source image format to be checked.
in	*bitDepth1*	First source image bit depth to be checked.
in	*format2*	Second source image format to be checked.
in	*bitDepth2*	Second source image bit depth to be checked.

3.38.2.7 bool HImageMerge::Merge (HImage * *srcImage*, HImage * *srcImage2*, HImage * *destImage*)

Merge two images.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*srcImage*	First source image to be merged.
in	*srcImage2*	Second source image to be merged.
in,out	*destImage*	Destination image.

3.38.2.8 bool HImageMerge::MultipleMerge (HImage ** *imageList*, int *size*, int *col*, int *row*, HImage * *destImage*)

Merge multiple images.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*imageList*	Array of images to be merged.
in	*size*	Number of images in the array.
in	*col*	Column count.
in	*row*	Row count.
in,out	*destImage*	Destination image.

3.38.2.9 bool HImageMerge::Stitch (HImage * *srcImage*, int *offsetSrcX*, int *offsetSrcY*, int *sizeSrcX*, int *sizeSrcY*, HImage * *destImage*, int *offsetDestX*, int *offsetDestY*)

Stitch source image to target image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*srcImage*	source image to be stitched.
in	*offsetSrcX*	offset x for source image ROI area of interest.
in	*offsetSrcY*	offset y for source image ROI area of interest.
in	*sizeSrcX*	size x for source image ROI area of interest.
in	*sizeSrcY*	size y for source image ROI area of interest.
in	*destImage*	target image to stitch.
in	*offsetDestX*	offset x for dest image to stitch.
in	*offsetDestY*	offset x for dest image to stitch.

3.39 HImageMirror Class Reference

Flip an image (horizontaly or verticaly).

```
#include <HImageMirror.h>
```

Public Member Functions

- void SetMirrorMode (eHMirror_Mode mode)

  Set the mirror mode.
- eHMirror_Mode GetMirrorMode ()

  Get the current mirror mode.
- bool CheckImageMirrorFormat (eHImageFormat format, int bitDepth)

  Check if a specific image format can be flipped.
- bool Mirror (HImage *srcImage, HImage *destImage)

  Flip an image.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.39.1 Detailed Description

Flip an image (horizontaly or verticaly).

3.39.2 Member Function Documentation

3.39.2.1 void HImageMirror::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.39.2.2 eHErrorCode HImageMirror::GetLastError ( )

See also

HGrabModule::GetLastError

3.39.2.3 void HImageMirror::SetMirrorMode (eHMirror_Mode *mode*)

Set the mirror mode.

Parameters

in	*mode*	The flip mode.

3.39.2.4 eHMirror_Mode HImageMirror::GetMirrorMode ( )

Get the current mirror mode.

Returns

The mirror mode.

3.39.2.5 bool HImageMirror::CheckImageMirrorFormat (eHImageFormat *format,* int *bitDepth*)

Check if a specific image format can be flipped.

Returns

True if an image with that format can be flipped, false otherwise.

Parameters

in	*format*	The image format.
in	*bitDepth*	The image bit depth.

3.39.2.6 bool HImageMirror::Mirror (HImage * *srcImage,* HImage * *destImage*)

Flip an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*srcImage*	The source image.
in,out	*destImage*	The flipped image.

3.40 HImageOverlay Class Reference

Write overlays on any supported image.

```
#include <HImageOverlay.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

MISC FUNCTIONS

- bool IsFormatSupported (eHImageFormat imageFormat)

  if the image format supports the overlay
- bool ForceGDIOverlay (bool enable)

  force to use GDI overlay mode

SETUP FUNCTIONS

- bool SetOverlayColor (COLORREF color)

  Set the color used to draw the overlays.
- COLORREF GetOverlayColor ()

  Retrieve the color used to draw the overlays.
- bool SetTextBackgroundColor (COLORREF color)

  Set the background color of the text overlays.
- COLORREF GetTextBackgroundColor ()

  Retrieve the color used in the background of text overlays.
- bool SetAlpha (int alpha)

  Set the alpha channel value (transparency) of the overlays.
- int GetAlpha ()

  Retrieve the alpha channel value used to draw the overlays.
- bool SetTextOverlayAlign (int horizontalAlign=-1, int verticalAlign=-1)

  Set the alignment of the text overlays on the image.
- bool SetTextOverlayFont (LPCTSTR fontName, bool bold=false)

  Set the font used for the text overlay.
- bool SetTextOverlaySize (int fontSize)

  Set the font size for text overlays.
- bool SetCrosshairThickness (unsigned int thickness)

  Set the line thickness of the crosshairs.
- unsigned int GetCrosshairThickness ()

  Get the line thickness of the crosshairs.
- bool SetLineThickness (unsigned int thickness)

  Set the line thickness (in pixels) for the functions that draw lines and shapes.
- unsigned int GetLineThickness ()

  Get the line thickness for the functions that draw lines and shapes.
- bool SetLineStyle (unsigned int style)

  Set the line style of the line overlays.
- bool SetLineColor (COLORREF color)

Set the line color of the line overlays.

- COLORREF GetLineColor ()

  Get the line color of the line overlays.
- UINT GetPolygonPixelCount (POINT pointsArray[ ], unsigned int pointsCount, HImage *image)

  Get the pixel count of a polygonal region "return The pixel count.

WRITE FUNCTIONS

- bool BeginDrawing (HImage *image)

  If many write functions are to be called for the same image, it is recommended to do it in a single pass.
- bool EndDrawing ()

  Burns all the overlays queued to be written since the BeginDrawing function was called.
- bool WriteAlignedTextOverlay (LPCTSTR text, HImage *image, RECT *resultRect=NULL)

  Write a text overlay on an image using the alignment set by SetTextOverlayAlign().
- bool WriteAlignedMultilinesTextOverlay (LPCTSTR text, HImage *image, RECT *resultRect=NULL)

  Obsolete, use WriteAlignedMultilinesTextOverlay instead.
- bool WriteTextOverlay (LPCTSTR text, POINT dest, HImage *image, RECT *resultRect=NULL)

  Write a text overlay on an image at the given position.
- bool WriteMultilineTextOverlay (LPCTSTR text, POINT dest, HImage *image, RECT *resultRect=NULL)

  Write a multiple-lines text overlay on an image at the given position.
- bool WriteLineOverlay (POINT origin, POINT dest, HImage *image)

  Draw a line overlay on an image.
- bool WriteFlatEndsLineOverlay (POINT origin, POINT dest, int endsLength, HImage *image)

  Draw a line overlay with a flat line on each end on an image.
- bool WriteRoundEndsLineOverlay (POINT origin, POINT dest, int radius, HImage *image)

  Draw a line overlay with large dot on each end on an image.
- bool WriteCrosshair (POINT dest, unsigned int sizeX, unsigned int sizeY, HImage *image)

  Draw a crosshair overlay on an image.
- bool WriteArrowOverlay (POINT origin, POINT dest, HImage *image)

  Draw an arrow overlay on an image.
- bool WriteRectOverlay (RECT rect, HImage *image)

  Draw a rectangle frame overlay on an image.
- bool WriteFilledRectOverlay (RECT rect, HImage *image)

  Draw a filled rectangle overlay on an image.
- bool WriteEllipseOverlay (RECT rect, HImage *image)

  Draw an ellipse overlay on an image.
- bool WriteFilledEllipseOverlay (RECT rect, HImage *image)

  Draw a filled ellipse overlay on an image.
- bool WriteCircleOverlay (POINT center, int radius, HImage *image)

  Draw a circle overlay on an image.
- bool WriteAngleOverlay (POINT pt1, POINT pt2, POINT pt3, bool clockwise, HImage *image)

  Draw an angle overlay on an image.
- bool WritePolyOverlay (POINT pointsArray[ ], unsigned int pointsCount, HImage *image)

  Draw a polygon overlay on an image.
- bool WriteFilledPolyOverlay (POINT pointsArray[ ], unsigned int pointsCount, HImage *image)

  Draw a filled polygon overlay on an image.
- bool WriteBezierOverlay (POINT pointsArray[ ], unsigned int pointsCount, HImage *image)

  Same as the WritePolyOverlay function except that point are conected with bezier lines instead of straight lines.
- bool WriteCustomOverlay (POINT pointsArray[ ], unsigned int pointsCount, HImage *image)

  Draw a custom overlay on an image.

3.40.1 Detailed Description

Write overlays on any supported image.

3.40.2 Member Function Documentation

3.40.2.1 void HImageOverlay::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.40.2.2 eHErrorCode HImageOverlay::GetLastError ( )

See also

HGrabModule::GetLastError

3.40.2.3 bool HImageOverlay::SetOverlayColor (COLORREF *color*)

Set the color used to draw the overlays.

If color and background color are the same, background will be transparent.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*color*	The color.

3.40.2.4 COLORREF HImageOverlay::GetOverlayColor ( )

Retrieve the color used to draw the overlays.

Returns

The color.

3.40.2.5 bool HImageOverlay::SetTextBackgroundColor (COLORREF *color*)

Set the background color of the text overlays.

If both the color and background color are the same, background will be transparent.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*color*	The color.

3.40.2.6 COLORREF HImageOverlay::GetTextBackgroundColor ( )

Retrieve the color used in the background of text overlays.

Returns

The color.

3.40.2.7 bool HImageOverlay::SetAlpha (int *alpha*)

Set the alpha channel value (transparency) of the overlays.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*alpha*	The alpha value from Full transparency(0) to Opaque(100).

3.40.2.8 int HImageOverlay::GetAlpha ( )

Retrieve the alpha channel value used to draw the overlays.

Returns

The alpha value from 0:Full transparency to 100:Opaque.

3.40.2.9 bool HImageOverlay::SetTextOverlayAlign (int *horizontalAlign* = `-1`, int *verticalAlign* = `-1`)

Set the alignment of the text overlays on the image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*horizontalAlign*	left aligned(-1), centered(0), right aligned(1).
in	*verticalAlign*	top aligned(-1), centered(0), bottom aligned(1).

3.40.2.10 bool HImageOverlay::SetTextOverlayFont (LPCTSTR *fontName*, bool *bold* = `false`)

Set the font used for the text overlay.

The font must be installed on the system.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*fontName*	The name of the font to use.
in	*bold*	True if font is bold.

3.40.2.11 bool HImageOverlay::SetTextOverlaySize (int *fontSize*)

Set the font size for text overlays.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*fontSize*	the font size in pts. Must be greater than 8

3.40.2.12 bool HImageOverlay::SetCrosshairThickness (unsigned int *thickness*)

Set the line thickness of the crosshairs.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*thickness*	The line thickness, in pixels.

3.40.2.13 unsigned int HImageOverlay::GetCrosshairThickness ( )

Get the line thickness of the crosshairs.

Returns

The crosshair thickness in pixels.

3.40.2.14 bool HImageOverlay::SetLineThickness (unsigned int *thickness*)

Set the line thickness (in pixels) for the functions that draw lines and shapes.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*thickness*	The line thickness, in pixels.

3.40.2.15 unsigned int HImageOverlay::GetLineThickness ( )

Get the line thickness for the functions that draw lines and shapes.

Returns

The line thickness in pixels.

3.40.2.16 bool HImageOverlay::SetLineStyle (unsigned int *style*)

Set the line style of the line overlays.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*style*	The line style. See Win32 function "ExtCreatePen" for possible style values.

3.40.2.17 bool HImageOverlay::SetLineColor (COLORREF *color*)

Set the line color of the line overlays.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*color*	The color.

3.40.2.18 COLORREF HImageOverlay::GetLineColor ( )

Get the line color of the line overlays.

Returns

The line color.

3.40.2.19 UINT HImageOverlay::GetPolygonPixelCount (POINT *pointsArray[ ],* unsigned int *pointsCount,* HImage * *image*)

Get the pixel count of a polygonal region "return The pixel count.

Parameters

in	*pointsArray*	An array of points that will be linked to form the polygon.
in	*pointsCount*	The number of points in the pointsArray.
in	*image*	The image on which the polygon would be burned

3.40.2.20 bool HImageOverlay::BeginDrawing (HImage * *image*)

If many write functions are to be called for the same image, it is recommended to do it in a single pass.

To do that, call BeginDrawing, then call all the WriteXXX functions to queue the writes. Finally, call EndDrawing to do burn all the overlays at once.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The target image on which to burn the text overlay

3.40.2.21 bool HImageOverlay::EndDrawing ( )

Burns all the overlays queued to be written since the BeginDrawing function was called.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.40.2.22 bool HImageOverlay::WriteAlignedTextOverlay (LPCTSTR *text*, HImage * *image*, RECT * *resultRect* = NULL)

Write a text overlay on an image using the alignment set by SetTextOverlayAlign().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*text*	The text string to burn on the image
in	*image*	The target image on which to burn the text overlay
out	*resultRect*	The area occupied by the text overlay.

3.40.2.23 bool HImageOverlay::WriteAlignedMultilinesTextOverlay (LPCTSTR *text*, HImage * *image*, RECT * *resultRect* = NULL)

Obsolete, use WriteAlignedMultilinesTextOverlay instead.

Write a multiple-lines text overlay on an image using the alignment set by SetTextOverlayAlign().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*text*	The text string to burn on the image
in	*image*	The target image on which to burn the text overlay
out	*resultRect*	The area occupied by the text overlay.

3.40.2.24 bool HImageOverlay::WriteTextOverlay (LPCTSTR *text*, POINT *dest*, HImage * *image*, RECT * *resultRect* = NULL)

Write a text overlay on an image at the given position.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*text*	The text string to burn on the image
in	*dest*	The pixel coordinates on which the text is left aligned.
in	*image*	The target image on which to burn the text overlay
out	*resultRect*	The area occupied by the text overlay.

3.40.2.25 bool HImageOverlay::WriteMultilineTextOverlay (LPCTSTR *text*, POINT *dest*, HImage * *image*, RECT * *resultRect* = NULL)

Write a multiple-lines text overlay on an image at the given position.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*text*	The text string to burn on the image
in	*dest*	The pixel coordinates on which the text is left aligned.
in	*image*	The target image on which to burn the text overlay
out	*resultRect*	The area occupied by the text overlay.

3.40.2.26 bool HImageOverlay::WriteLineOverlay (POINT *origin*, POINT *dest*, HImage * *image*)

Draw a line overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*origin*	The coordinates of the origin pixel.
in	*dest*	The coordinates of the destination pixel.
in	*image*	The target image on which to burn the overlay

3.40.2.27 bool HImageOverlay::WriteFlatEndsLineOverlay (POINT *origin,* POINT *dest,* int *endsLength,* HImage * *image*)

Draw a line overlay with a flat line on each end on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*origin*	The coordinates of the origin pixel.
in	*dest*	The coordinates of the destination pixel.
in	*endsLength*	The length of each flat lines, in pixel.
in	*image*	The target image on which to burn the overlay

3.40.2.28 bool HImageOverlay::WriteRoundEndsLineOverlay (POINT *origin,* POINT *dest,* int *radius,* HImage * *image*)

Draw a line overlay with large dot on each end on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*origin*	The coordinates of the origin pixel.
in	*dest*	The coordinates of the destination pixel.
in	*radius*	The radius of the dots, in pixels.
in	*image*	The target image on which to burn the overlay

3.40.2.29 bool HImageOverlay::WriteCrosshair (POINT *dest,* unsigned int *sizeX,* unsigned int *sizeY,* HImage * *image*)

Draw a crosshair overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*dest*	The coordinates of the crossahir's center.
in	*sizeX*	The width of the crosshair.
in	*sizeY*	The height of the crosshair.
in	*image*	The target image on which to burn the overlay

3.40.2.30 bool HImageOverlay::WriteArrowOverlay (POINT *origin,* POINT *dest,* HImage * *image*)

Draw an arrow overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code

Parameters

in	*origin*	The coordinates of the origin pixel.
in	*dest*	The coordinates of the destination pixel. The arrow will point here.
in	*image*	The target image on which to burn the overlay

3.40.2.31 bool HImageOverlay::WriteRectOverlay (RECT *rect,* HImage * *image*)

Draw a rectangle frame overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*rect*	A RECT structure to pass the pixels coordinates that define the boundaries.
in	*image*	The target image on which to burn the overlay

3.40.2.32 bool HImageOverlay::WriteFilledRectOverlay (RECT *rect,* HImage * *image*)

Draw a filled rectangle overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*rect*	A RECT structure to pass the pixels coordinates that define the boundaries.
in	*image*	The target image on which to burn the overlay

3.40.2.33 bool HImageOverlay::WriteEllipseOverlay (RECT *rect,* HImage * *image*)

Draw an ellipse overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*rect*	A RECT structure to pass the pixels coordinates that define the horizontal and vertical boundaries.
in	*image*	The target image on which to burn the overlay

3.40.2.34 bool HImageOverlay::WriteFilledEllipseOverlay (RECT *rect,* HImage * *image*)

Draw a filled ellipse overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*rect*	A RECT structure to pass the pixels coordinates that define the horizontal and vertical boundaries.
in	*image*	The target image on which to burn the overlay.

3.40.2.35 bool HImageOverlay::WriteCircleOverlay (POINT *center,* int *radius,* HImage * *image*)

Draw a circle overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*center*	The coordinates of the pixel at the center.
in	*radius*	The radius (in pixel) of the circle.
in	*image*	The target image on which to burn the overlay.

3.40.2.36 bool HImageOverlay::WriteAngleOverlay (POINT *pt1,* POINT *pt2,* POINT *pt3,* bool *clockwise,* HImage * *image*)

Draw an angle overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pt1*	The coordinates of the first pixel.
in	*pt2*	The coordinates of the second pixel (where the angle is computed).
in	*pt3*	The coordinates of the third pixel.
in	*clockwise*	True if the angle computed is the one clockwise from pt1 to pt3. False for anti-clcokwise.
in	*image*	The target image on which to burn the overlay.

3.40.2.37 bool HImageOverlay::WritePolyOverlay (POINT *pointsArray[ ],* unsigned int *pointsCount,* HImage * *image*)

Draw a polygon overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pointsArray*	An array of points that will be linked to form the polygon.
in	*pointsCount*	The number of points in the pointsArray.
in	*image*	The target image on which to burn the overlay.

3.40.2.38 bool HImageOverlay::WriteFilledPolyOverlay (POINT *pointsArray[ ],* unsigned int *pointsCount,* HImage * *image*)

Draw a filled polygon overlay on an image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pointsArray*	An array of points that will be linked to form the polygon.
in	*pointsCount*	The number of points in the pointsArray.
in	*image*	The target image on which to burn the overlay.

3.40.2.39 bool HImageOverlay::WriteBezierOverlay (POINT *pointsArray[ ],* unsigned int *pointsCount,* HImage * *image*)

Same as the WritePolyOverlay function except that point are conected with bezier lines instead of straight lines.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pointsArray*	An array of points that will be linked to form the bezier overlay.
in	*pointsCount*	The number of points in the pointsArray.
in	*image*	The target image on which to burn the overlay.

3.40.2.40 bool HImageOverlay::WriteCustomOverlay (POINT *pointsArray[ ],* unsigned int *pointsCount,* HImage * *image*)

Draw a custom overlay on an image.

Every pixels in the pointsArray will be colored using the overlay color. The pixels are not automatically linked to each other.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pointsArray*	An array of points that will make the overlay.
in	*pointsCount*	The number of points in the pointsArray.
in	*image*	The target image on which to burn the overlay.

3.41 HImageResize Class Reference

Changes an image size using different interpolation algorithms.

```
#include <HImageResize.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

SETUP FUNCTIONS

- void SetSourceRoi (int offsetX, int offsetY, int sizeX, int sizeY)
  Set the region of interest on the source image.
- void GetSourceRoi (int &offsetX, int &offsetY, int &sizeX, int &sizeY)
  Get the region of interest on the source image.
- void SetDestinationSize (int sizeX, int sizeY)
  Set size of the output image.
- void GetDestinationSize (int &sizeX, int &sizeY)
  Get the size of the output image.
- void SetKeepImageRatio (bool ratio)
  Tell the resize function to keep the aspect ratio of the original image.
- bool IsKeepImageRatio ()
  Check if the resize functions try to keep the original aspect ratio.
- void SetInterpolationMode (eHResizeInterpolationMode mode)
  Set the interpolation mode used for image resizing.
- eHResizeInterpolationMode GetInterpolationMode ()
  Get the interpolation mode.

RESIZE/RESAMPLE FUNCTIONS

- bool CheckImageResizeFormat (eHImageFormat format, int bitDepth)
  Check if the image format and bit depth are supported by the image resize function.
- bool Resize (HImage *srcImage, HImage *destImage)
  Perform the image resize from srcImage to destImage.
- bool CheckImageReSampleFormat (eHImageFormat format, int bitDepth)
  Check if the image format and bit depth are supported by the image resample function.
- bool ReSample (int factor, HImage *srcImage, HImage *destImage)
  Perform the image sampling from srcImage to destImage.

3.41.1 Detailed Description

Changes an image size using different interpolation algorithms.

3.41.2 Member Function Documentation

3.41.2.1 void HImageResize::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.41.2.2 eHErrorCode HImageResize::GetLastError ( )

See also

HGrabModule::GetLastError

3.41.2.3 void HImageResize::SetSourceRoi (int *offsetX*, int *offsetY*, int *sizeX*, int *sizeY*)

Set the region of interest on the source image.

Parameters

in	*offsetX*	The horizontal offset of the region.
in	*offsetY*	The vertical offset of the region.
in	*sizeX*	The width of the region.
in	*sizeY*	The height of the region.

3.41.2.4 void HImageResize::GetSourceRoi (int & *offsetX,* int & *offsetY,* int & *sizeX,* int & *sizeY*)

Get the region of interest on the source image.

Parameters

out	*offsetX*	The horizontal offset of the region.
out	*offsetY*	The vertical offset of the region.
out	*sizeX*	The width of the region.
out	*sizeY*	The height of the region.

3.41.2.5 void HImageResize::SetDestinationSize (int *sizeX,* int *sizeY*)

Set size of the output image.

Parameters

in	*sizeX*	The width of the image.
in	*sizeY*	The height of the image.

3.41.2.6 void HImageResize::GetDestinationSize (int & *sizeX,* int & *sizeY*)

Get the size of the output image.

Parameters

out	*sizeX*	The width of the output image.
out	*sizeY*	The height of the output image.

3.41.2.7 void HImageResize::SetKeepImageRatio (bool *ratio*)

Tell the resize function to keep the aspect ratio of the original image.

Parameters

in	*ratio*	True to keep the aspect ratio, false to resize to the full destination size.

3.41.2.8 bool HImageResize::IsKeepImageRatio ( )

Check if the resize functions try to keep the original aspect ratio.

Returns

True if the aspect ratio is kept, false otherwise.

3.41.2.9 void HImageResize::SetInterpolationMode (eHResizeInterpolationMode *mode*)

Set the interpolation mode used for image resizing.

Parameters

in	*mode*	The interpolation mode.

3.41.2.10 eHResizeInterpolationMode HImageResize::GetInterpolationMode ( )

Get the interpolation mode.

Returns

The interpolation mode in use.

3.41.2.11 bool HImageResize::CheckImageResizeFormat (eHImageFormat *format,* int *bitDepth*)

Check if the image format and bit depth are supported by the image resize function.

Returns

True if the image format and bit depth combination is supported. False otherwise.

Parameters

in	*format*	Source image format.
in	*bitDepth*	Source image bit depth.

3.41.2.12 bool HImageResize::Resize (HImage * *srcImage,* HImage * *destImage*)

Perform the image resize from srcImage to destImage.

Returns

Return true if the resize was successful, false otherwise.

Parameters

in	*srcImage*	Source image to be resized
out	*destImage*	Resized destination image

3.41.2.13 bool HImageResize::CheckImageReSampleFormat (eHImageFormat *format,* int *bitDepth*)

Check if the image format and bit depth are supported by the image resample function.

Returns

True if the image format and bit depth combination is supported. False otherwise.

Parameters

in	*format*	Source image format to be check
in	*bitDepth*	Source image bit depth to be check

3.41.2.14 bool HImageResize::ReSample (int *factor*, HImage * *srcImage*, HImage * *destImage*)

Perform the image sampling from srcImage to destImage.

Returns

Return true if the resampling was successful, false otherwise.

Parameters

in	*factor*	sampling factor (1:X)
in	*srcImage*	source image to be resampled
out	*destImage*	resampled destination image

3.42 HIODriver Class Reference

Manage an I/O port.

```
#include <HIODriver.h>
```

Public Member Functions

- HIODriver (LPCTSTR filePath)
  Load an external IO driver.
- bool GetDriverName (LPTSTR name)
  Get the name of the driver managing the IO devices.

INPUT FUNCTIONS

- bool InputDevicePresent ()
  Check if an input device is available and connected.
- int GetInputDeviceCount ()
  Get the number of input devices managed by this driver.
- bool GetInputDeviceName (int deviceIndex, LPTSTR name)
  Get the name of an input device.
- bool InitializeDeviceInput (int deviceIndex)
  Initialize the device driver needed for input operations.
- bool UnInitializeDeviceInput (int deviceIndex)
  Uninitialize the input device driver.
- int GetDeviceInputLineCount (int deviceIndex)
  Get the number of input lines available on an input device.
- bool GetDeviceInputLineInfo (int deviceIndex, int lineIndex, int *ID, LPTSTR name)
  Get information on each available input line.
- int GetPollingDelay (int deviceIndex, int lineID)
  Get the polling delay for a specific input.
- eHInputLevel GetCurrentLevel (int deviceIndex, int lineID)
  Get the last known state of an input line.
- bool Monitor (int deviceIndex, int lineID, InputSupportCallback callback, int pollingDelay=100, int debouncerDelay=0, ULONG_PTR userData=0, void *userPtr=NULL)
  Start monitoring level changes on a specific input line.

OUTPUT FUNCTIONS

- bool OutputDevicePresent ()
  Check if the output device is available and connected.
- int GetOutputDeviceCount ()

Get the number of output devices managed by this driver.
- bool GetOutputDeviceName (int deviceIndex, LPTSTR name)

  Get the name of an output device.
- bool InitializeDeviceOutput (int deviceIndex)

  Initialize the device drivers needed to do output operations.
- bool UnInitializeDeviceOutput (int deviceIndex)

  Uninitializes the output device driver.
- int GetDeviceOutputLineCount (int deviceIndex)

  Get the number of output lines available on an output device.
- bool GetDeviceOutputLineInfo (int deviceIndex, int index, int ∗ID, LPTSTR name)

  Get information on each available output line.
- eHOutputLevel GetOutputLevel (int deviceIndex, int lineID)

  Get the last known state of an output line.
- bool SetOutputLevel (int deviceIndex, int lineID, eHOutputLevel outputLevel)

  Set the state of an output line.

PULSE GENERATOR FUNCTIONS

- bool PulseGeneratorPresent ()

  Check if a pulse generator is available for the current grabber.
- bool InitializePulseGenerator ()

  Initialize the pulse generator.
- bool UnInitializePulseGenerator ()

  Uninitialize the pulse generator.
- bool StartPulseGenerator (double freq, double activeDuration)

  Start the pulse generator at a specific frequency.
- bool StopPulseGenerator ()

  Stop the pulse generator.

DATA ACQUISITION (DAQ) FUNCTIONS

- bool DaqPresent ()

  Check if a DAQ device is available.
- int GetDaqDeviceCount ()

  Get the number of DAQ devices managed by this driver.
- bool GetDaqDeviceName (int deviceIndex, LPTSTR name)

  Get the name of a DAQ device.
- bool InitializeDaqDevice (int deviceIndex)

  Initialize the device driver needed to do DAQ operations.
- bool UnInitializeDaqDevice (int deviceIndex)

  Uninitialize the DAQ device driver.
- bool SetTimeSourceDLL (int deviceIndex, LPCTSTR dllPath, int timeSourceOffsetUS)

  Set the time source dll that will be used to time stamp each sample.
- int GetDaqDeviceLineCount (int deviceIndex)

  Get the number of lines available on the DAQ device.
- bool GetDaqDeviceLineInfo (int deviceIndex, int index, int ∗ID, LPTSTR name)

  Get information on each available DAQ line.
- int GetDaqDeviceLineRangeSettingsCount (int deviceIndex)

  Get the number of various input range available with the DAQ device.
- bool GetDaqDeviceLineRangeSettingsInfo (int deviceIndex, int index, int ∗ID, LPTSTR name, double ∗min, double ∗max)

  Get information on each available DAQ input range.
- int GetMaxSampleRate (int deviceIndex, int lineCount, int ∗idMap)

  Get maximum sample rate frequency.
- bool StartCapture (int deviceIndex, int speed, double pollingTime, int lineMapCount, int ∗lineMap, int range, CDaqSupportCallback ∗callback)

  Start DAQ capture.
- bool StopCapture (int deviceIndex)

  Stop Capture DAQ data.

- bool GetSample (int deviceIndex, long timeStamp, unsigned short timeStampMS, unsigned short time↩StampUS, HDaqDataSet *daqSet)
  Retrieve a dataset from a specified time stamp.
- int GetDaqDeviceParametersCount (int deviceIndex)
  Get the number of parameter the DAQ Device is supporting.
- bool GetDaqDeviceParametersInfo (int deviceIndex, int index, eDAQDeviceParamType *type, LPTSTR name, LPTSTR desc)
  Retreive Device parameter information.
- int GetDaqDeviceEnumParameterItemCount (int deviceIndex, int index)
  Get the number of enumeration available for a specific parameter item.
- bool GetDaqDeviceEnumParameterItemInfo (int deviceIndex, int index, int itemIndex, LPTSTR name, int *value)
  Retreive the enumeration value and name for the specified item index.
- bool GetDaqDeviceFloatParameterItemInfo (int deviceIndex, int index, double *minV, double *maxV)
  Retreive floating point type parameter values for the specified item index.
- bool GetDaqDeviceIntParameterItemInfo (int deviceIndex, int index, long *minV, long *maxV)
  Retreive integer type parameter values for the specified item index.
- bool GetDaqDeviceEnumParameterValue (int deviceIndex, int index, int *value)
  Get current enumeration type parameter value.
- bool SetDaqDeviceEnumParameterValue (int deviceIndex, int index, int value)
  Set enumeration type parameter value.
- bool GetDaqDeviceFloatParameterValue (int deviceIndex, int index, double *value)
  Get current float type parameter value.
- bool SetDaqDeviceFloatParameterValue (int deviceIndex, int index, double value)
  Set float type parameter value.
- bool GetDaqDeviceIntParameterValue (int deviceIndex, int index, long *value)
  Get current integer type parameter value.
- bool SetDaqDeviceIntParameterValue (int deviceIndex, int index, long value)
  Set float type parameter value.

3.42.1 Detailed Description

Manage an I/O port.

This class allows to monitor the input/output line(s) of a grabber or any other supported external IO device. Call the HGrabModule::GetIODriver to access the inputs of the grabber or use the HIODriver constructor to load the .dll of a supported external IO device.

3.42.2 Constructor & Destructor Documentation

3.42.2.1 HIODriver::HIODriver (LPCTSTR *filePath*)

Load an external IO driver.

Parameters

in	*filePath*	The file path of the .dll to load. The supported .dll are located in the Hermes/IO folder.

3.42.3 Member Function Documentation

3.42.3.1 bool HIODriver::GetDriverName (LPTSTR *name*)

Get the name of the driver managing the IO devices.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*name*	The driver's name. Allocate TCHAR[MAX_PATH].

3.42.3.2 bool HIODriver::InputDevicePresent ( )

Check if an input device is available and connected.

Make sure the device is present before calling any other input functions.

Returns

True if input can be used.

3.42.3.3 int HIODriver::GetInputDeviceCount ( )

Get the number of input devices managed by this driver.

Normally, this will be 1 for grabber managed I/O device, but could be more for dedicated external I/O devices.

Returns

The input device count.

3.42.3.4 bool HIODriver::GetInputDeviceName (int *deviceIndex*, LPTSTR *name*)

Get the name of an input device.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1].
in,out	*name*	The device's name. Allocate TCHAR[MAX_PATH].

3.42.3.5 bool HIODriver::InitializeDeviceInput (int *deviceIndex*)

Initialize the device driver needed for input operations.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1].

3.42.3.6 bool HIODriver::UnInitializeDeviceInput (int *deviceIndex*)

Uninitialize the input device driver.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1].

3.42.3.7 int HIODriver::GetDeviceInputLineCount (int *deviceIndex*)

Get the number of input lines available on an input device.

Returns

The input line count.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1].

3.42.3.8 bool HIODriver::GetDeviceInputLineInfo (int *deviceIndex*, int *lineIndex*, int * *ID*, LPTSTR *name*)

Get information on each available input line.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1].
in	*lineIndex*	The line index. [0, GetInputLineCount()-1].
out	*ID*	The lineID used to identify the input line.
out	*name*	The name of the input line. Allocate TCHAR [MAX_PATH].

3.42.3.9 int HIODriver::GetPollingDelay (int *deviceIndex*, int *lineID*)

Get the polling delay for a specific input.

Returns

the polling delay in milliseconds.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1].
in	*lineID*	The lineID of the input line to query.

3.42.3.10 eHInputLevel HIODriver::GetCurrentLevel (int *deviceIndex*, int *lineID*)

Get the last known state of an input line.

It can be 'level-high', 'level-low' or in some cases 'unknown'.

Returns

The last known state of the input line.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1].
in	*lineID*	The lineID of the input line to query.

3.42.3.11 bool HIODriver::Monitor (int *deviceIndex*, int *lineID*, InputSupportCallback *callback*, int *pollingDelay* = 100, int *debouncerDelay* = 0, ULONG_PTR *userData* = 0, void * *userPtr* = NULL)

Start monitoring level changes on a specific input line.

If "callback" is NULL, stop monitoring the line. If the device can't monitor more than one line at the time, this will override the older monitoring callback. If polling delay is 0, we try using the interrupt mode if available.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1].
in	*lineID*	The lineID of the input line to monitor.
in	*callback*	The callback function. To stop monitoring the line, set the callback to NULL.
in	*pollingDelay*	This value determines the frequency, in milliseconds, at which the line will be queried. Using a low value will require more resources while using a high value could omit edges.
in	*debouncerDelay*	The debouncer delay (in ms). Useful for analog devices.
in	*userData*	This value will be sent back in the callback function.
in	*userPtr*	This pointer will be sent back in the callback function.

3.42.3.12 bool HIODriver::OutputDevicePresent ( )

Check if the output device is available and connected.

Make sure the device is present before calling any other output functions in this class.

Returns

True if the output can be used.

3.42.3.13 int HIODriver::GetOutputDeviceCount ( )

Get the number of output devices managed by this driver.

Normally, this will be 1 for grabber managed I/O devices, but could be more for dedicated external I/O devices.

Returns

The output device count.

3.42.3.14 bool HIODriver::GetOutputDeviceName (int *deviceIndex*, LPTSTR *name*)

Get the name of an output device.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetOutputDeviceCount()-1].
in,out	*name*	The device's name. Allocate TCHAR[MAX_PATH].

3.42.3.15 bool HIODriver::InitializeDeviceOutput (int *deviceIndex*)

Initialize the device drivers needed to do output operations.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetOutputDeviceCount()-1].

3.42.3.16 bool HIODriver::UnInitializeDeviceOutput (int *deviceIndex*)

Uninitializes the output device driver.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetOutputDeviceCount()-1].

3.42.3.17 int HIODriver::GetDeviceOutputLineCount (int *deviceIndex*)

Get the number of output lines available on an output device.

Returns

The output line count.

Parameters

in	*deviceIndex*	The device index. [0, GetOutputDeviceCount()-1].

3.42.3.18 bool HIODriver::GetDeviceOutputLineInfo (int *deviceIndex*, int *index*, int * *ID*, LPTSTR *name*)

Get information on each available output line.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetInputDeviceCount()-1] .
in	*index*	The line index. [0, GetOutputLineCount()-1].
out	*ID*	The lineID used to identify the output line.
out	*name*	The name of the output line. Allocate TCHAR [MAX_PATH].

3.42.3.19 eHOutputLevel HIODriver::GetOutputLevel (int *deviceIndex*, int *lineID*)

Get the last known state of an output line.

It can be 'level-high', 'level-low' or in some cases 'unknown'.

Returns

The last known state of the output line.

Parameters

in	*deviceIndex*	The device index. [0, GetOutputDeviceCount()-1]
in	*lineID*	The lineID of the output line to query.

3.42.3.20 bool HIODriver::SetOutputLevel (int *deviceIndex*, int *lineID*, eHOutputLevel *outputLevel*)

Set the state of an output line.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetOutputDeviceCount()-1].
in	*lineID*	The lineID of the output line.
in	*outputLevel*	The desired output state.

3.42.3.21 bool HIODriver::PulseGeneratorPresent ( )

Check if a pulse generator is available for the current grabber.

Make sure the pulse generator is present before calling any other function related to it.

Returns

True if pulse generator can be used.

3.42.3.22 bool HIODriver::InitializePulseGenerator ( )

Initialize the pulse generator.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.42.3.23 bool HIODriver::UnInitializePulseGenerator ( )

Uninitialize the pulse generator.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.42.3.24 bool HIODriver::StartPulseGenerator (double *freq*, double *activeDuration*)

Start the pulse generator at a specific frequency.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*freq*	The number of pulses per second.
in	*activeDuration*	The pulse duration.

3.42.3.25 bool HIODriver::StopPulseGenerator ( )

Stop the pulse generator.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.42.3.26 bool HIODriver::DaqPresent ( )

Check if a DAQ device is available.

Make sure the device is available before calling any other DAQ function.

Returns

True if DAQ can be used.

3.42.3.27 int HIODriver::GetDaqDeviceCount ( )

Get the number of DAQ devices managed by this driver.

Normally, this will be 1 for grabber managed I/O Drivers, but could be more for dedicated external I/O devices.

Returns

The DAQ device count.

3.42.3.28 bool HIODriver::GetDaqDeviceName (int *deviceIndex*, LPTSTR *name*)

Get the name of a DAQ device.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetDaqDeviceCount()-1].
in,out	*name*	The device's name. Allocate TCHAR[MAX_PATH].

3.42.3.29 bool HIODriver::InitializeDaqDevice (int *deviceIndex*)

Initialize the device driver needed to do DAQ operations.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetDaqDeviceCount()-1].

3.42.3.30 bool HIODriver::UnInitializeDaqDevice (int *deviceIndex*)

Uninitialize the DAQ device driver.

Parameters

in	*deviceIndex*	The device index. [0, GetDaqDeviceCount()-1].

3.42.3.31 bool HIODriver::SetTimeSourceDLL (int *deviceIndex*, LPCTSTR *dllPath*, int *timeSourceOffsetUS*)

Set the time source dll that will be used to time stamp each sample.

If this function isn't used, DAQ time srtamping will default to the system clock.

Parameters

in	*deviceIndex*	The device index. [0, GetDaqDeviceCount()-1].
in	*dllPath*	The file path of the .dll to load.
in	*timeSource↩ OffsetUS*	The time offset, in milliseconds, to apply (if any).

3.42.3.32 int HIODriver::GetDaqDeviceLineCount (int *deviceIndex*)

Get the number of lines available on the DAQ device.

Returns

The DAQ line count.

Parameters

in	*deviceIndex*	The device index. [0, GetDaqDeviceCount()-1].

3.42.3.33 bool HIODriver::GetDaqDeviceLineInfo (int *deviceIndex*, int *index*, int * *ID*, LPTSTR *name*)

Get information on each available DAQ line.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetDaqDeviceCount()-1] .
in	*index*	The line index.
out	*ID*	The lineID used to identify the DAQ line.
out	*name*	The name of the DAQ line. Allocate TCHAR [MAX_PATH].

3.42.3.34 int HIODriver::GetDaqDeviceLineRangeSettingsCount (int *deviceIndex*)

Get the number of various input range available with the DAQ device.

Returns

The DAQ range count.

Parameters

in	*deviceIndex*	The device index. [0, GetDaqDeviceCount()-1].

3.42.3.35 bool HIODriver::GetDaqDeviceLineRangeSettingsInfo (int *deviceIndex*, int *index*, int * *ID*, LPTSTR *name*, double * *min*, double * *max*)

Get information on each available DAQ input range.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index. [0, GetDaqDeviceCount()-1].
in	*index*	The range index.
out	*ID*	Range ID.
out	*name*	Range name.
out	*min*	Range minimum value.
out	*max*	Range maximum value.

3.42.3.36 int HIODriver::GetMaxSampleRate (int *deviceIndex*, int *lineCount*, int * *idMap*)

Get maximum sample rate frequency.

The maximum sample rate frequency may be limited by the selected lines and device line read mode. Refer to the hardware manual.

Returns

return the maximum sample rate frequency

Parameters

in	*deviceIndex*	The device index.
in	*lineCount*	The number of lines to read.
in	*idMap*	line ID array to read.

3.42.3.37 bool HIODriver::StartCapture (int *deviceIndex,* int *speed,* double *pollingTime,* int *lineMapCount,* int ∗ *lineMap,* int *range,* **CDaqSupportCallback** ∗ *callback*)

Start DAQ capture.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index.
in	*speed*	Sample rate.
in	*pollingTime*	Polling time.
in	*lineMapCount*	The number of lines to read.
in	*lineMap*	line ID array tp read.
in	*range*	range ID.
in	*callback*	capture callback function.

3.42.3.38 bool HIODriver::StopCapture (int *deviceIndex*)

Stop Capture DAQ data.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index.

3.42.3.39 bool HIODriver::GetSample (int *deviceIndex,* long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS,* **HDaqDataSet** ∗ *daqSet*)

Retrieve a dataset from a specified time stamp.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	The device index.
in	*timeStamp*	time stamp.
in	*timeStampMS*	time stamp.
in	*timeStampUS*	time stamp.
out	*daqSet*	DAQ data.

3.42.3.40 int HIODriver::GetDaqDeviceParametersCount (int *deviceIndex*)

Get the number of parameter the DAQ Device is supporting.

Returns

number of parameter

Parameters

in	*deviceIndex*	device ID.

3.42.3.41 bool HIODriver::GetDaqDeviceParametersInfo (int *deviceIndex,* int *index,* eDAQDeviceParamType * *type,* LPTSTR *name,* LPTSTR *desc*)

Retreive Device parameter information.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	index of parameter to query.
out	*type*	parameter type.
out	*name*	parameter name.
out	*desc*	parameter description.

3.42.3.42 int HIODriver::GetDaqDeviceEnumParameterItemCount (int *deviceIndex,* int *index*)

Get the number of enumeration available for a specific parameter item.

Returns

number of enumeration items

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index.

3.42.3.43 bool HIODriver::GetDaqDeviceEnumParameterItemInfo (int *deviceIndex,* int *index,* int *itemIndex,* LPTSTR *name,* int * *value*)

Retreive the enumeration value and name for the specified item index.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index.
in	*itemIndex*	enumeration item index to query.
out	*name*	enumeration item name.
out	*value*	enumeration item value.

3.42.3.44 bool HIODriver::GetDaqDeviceFloatParameterItemInfo (int *deviceIndex*, int *index*, double * *minV*, double * *maxV*)

Retreive floating point type parameter values for the specified item index.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index to query.
out	*minV*	minimum value.
out	*maxV*	maximum value.

3.42.3.45 bool HIODriver::GetDaqDeviceIntParameterItemInfo (int *deviceIndex*, int *index*, long * *minV*, long * *maxV*)

Retreive integer type parameter values for the specified item index.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index to query.
out	*minV*	minimum value.
out	*maxV*	maximum value.

3.42.3.46 bool HIODriver::GetDaqDeviceEnumParameterValue (int *deviceIndex*, int *index*, int * *value*)

Get current enumeration type parameter value.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index.
out	*value*	current value.

3.42.3.47 bool HIODriver::SetDaqDeviceEnumParameterValue (int *deviceIndex*, int *index*, int *value*)

Set enumeration type parameter value.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index.
in	*value*	the value to put.

3.42.3.48 bool HIODriver::GetDaqDeviceFloatParameterValue (int *deviceIndex*, int *index*, double * *value*)

Get current float type parameter value.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index.
out	*value*	current value.

3.42.3.49 bool HIODriver::SetDaqDeviceFloatParameterValue (int *deviceIndex*, int *index*, double *value*)

Set float type parameter value.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index.
in	*value*	the value to put.

3.42.3.50 bool HIODriver::GetDaqDeviceIntParameterValue (int *deviceIndex*, int *index*, long * *value*)

Get current integer type parameter value.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index.
out	*value*	current value.

3.42.3.51 bool HIODriver::SetDaqDeviceIntParameterValue (int *deviceIndex*, int *index*, long *value*)

Set float type parameter value.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceIndex*	device ID.
in	*index*	parameter index.
in	*value*	the value to put.

3.43 HItem Class Reference

This is a utility class used by some other classes.

`#include <HItem.h>`

Public Member Functions

- int GetID ()

  Get the item ID value.
- LPCTSTR GetDescription ()

  Get the item description.

3.43.1 Detailed Description

This is a utility class used by some other classes.

3.43.2 Member Function Documentation

3.43.2.1 int HItem::GetID ( )

Get the item ID value.

Returns

The ID.

3.43.2.2 LPCTSTR HItem::GetDescription ( )

Get the item description.

Returns

The description.

3.44 HJpegCompression Class Reference

This class JPEG compress or decompress an HImage This class can be used to jpeg compress/decompress.

```
#include <HJpegCompression.h>
```

Public Member Functions

- bool IsFormatSupported (eHImageFormat format) const

  Check if the image format is supported by the Encode function.
- bool Encode (HImage *imageIn, HImage *imageOut, int quality)

  Jpeg compression.
- bool Decode (HImage *imageIn, HImage *imageOut)

  Jpeg decompression.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.44.1 Detailed Description

This class JPEG compress or decompress an HImage This class can be used to jpeg compress/decompress.

3.44.2 Member Function Documentation

3.44.2.1 void HJpegCompression::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.44.2.2 eHErrorCode HJpegCompression::GetLastError ( )

See also

HGrabModule::GetLastError

3.44.2.3 bool HJpegCompression::IsFormatSupported (eHImageFormat *format*) const

Check if the image format is supported by the Encode function.

Returns

True if the image format is supported, false otherwise.

Parameters

in	*format*	Image format to be checked.

3.44.2.4 bool HJpegCompression::Encode (HImage * *imageIn*, HImage * *imageOut*, int *quality*)

Jpeg compression.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*imageIn*	Source image to be compress
in	*imageOut*	output image compressed.
in	*quality*	Compress quality(1-100).

3.44.2.5 bool HJpegCompression::Decode (HImage * *imageIn*, HImage * *imageOut*)

Jpeg decompression.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*imageIn*	Source image to be decompress
in	*imageOut*	output image decompressed.

3.45 HLineScanManager Class Reference

This class is used for managing images originated from a line scan sensor.

```
#include <HLineScanManager.h>
```

Public Member Functions

- void Configure (HImage *image, int lineCount)

  Configure the line scan renderring.
- void SetLineCount (int lineCount)

  Set the number of total lines to be displayed.
- void SetDirection (int order, int orientation, bool jump)

  Set the waterfall flow direction. Depends on scan direction.
- void PushImageToQueue (HImage *image)

  Add image to the line scan manager FIFO queue of images.
- bool GetLiveImage (HImage *lineScanImage)

  Build a resulting image from a live stream source.
- bool GetImageFromSeq (HImage *lineScanImage, HSequence *sequence, int indexPosition)

  Build a resulting image from a previously recorded sequence file.

- bool ConfigureImageExport (int digits, eHJpegQuality jpegQuality, bool multiTIFF)

  Configure the line scan manager for exporting a waterfall renderred image.
- bool ConfigureAVIExport (int videoCodec, int audioSource, int audioCodec)

  Configure the line scan manager for exporting a waterfall renderred AVI.
- bool ConfigureMOVExport (int videoCodec, int audioSource, int videoQuality, unsigned long audioSample↩ Rate, short audioSampleSize, short audioChannelCount, unsigned long audioCodecType)

  Configure the line scan manager for exporting a waterfall renderred MOV (32 bit only).
- bool Export (HSequence ∗sequence, int destImageHeight, int startFrame, int endFrame, eHExportFormat fileFormat, LPCTSTR destName, int rotationAngle=0, double frameRate=30)

  Export into a waterfall renderred sequence file.
- int GetCurrentExportedFrame ()

  Get the image currently being exported.
- bool GetTimeStampImages (HImage ∗top, HImage ∗center, HImage ∗bottom)

  Retreive reference images for displaying timestamps on the resulting line scan image.
- void Configure (HImage ∗image, int lineCount)

  Configure the line scan renderring.
- void SetLineCount (int lineCount)

  Set the number of total lines to be displayed.
- void SetDirection (int order, int orientation, bool jump)

  Set the waterfall flow direction. Depends on scan direction.
- void PushImageToQueue (HImage ∗image)

  Add image to the line scan manager FIFO queue of images.
- bool GetLiveImage (HImage ∗lineScanImage)

  Build a resulting image from a live stream source.
- bool GetImageFromSeq (HImage ∗lineScanImage, HSequence ∗sequence, int indexPosition)

  Build a resulting image from a previously recorded sequence file.
- bool ConfigureImageExport (int digits, eHJpegQuality jpegQuality, bool multiTIFF)

  Configure the line scan manager for exporting a waterfall renderred image.
- bool ConfigureAVIExport (int videoCodec, int audioSource, int audioCodec)

  Configure the line scan manager for exporting a waterfall renderred AVI.
- bool ConfigureMOVExport (int videoCodec, int audioSource, int videoQuality, unsigned long audioSample↩ Rate, short audioSampleSize, short audioChannelCount, unsigned long audioCodecType)

  Configure the line scan manager for exporting a waterfall renderred MOV (32 bit only).
- bool Export (HSequence ∗sequence, int destImageHeight, int startFrame, int endFrame, eHExportFormat fileFormat, LPCTSTR destName, int rotationAngle=0, double frameRate=30)

  Export into a waterfall renderred sequence file.
- int GetCurrentExportedFrame ()

  Get the image currently being exported.
- bool GetTimeStampImages (HImage ∗top, HImage ∗center, HImage ∗bottom)

  Retreive reference images for displaying timestamps on the resulting line scan image.

3.45.1 Detailed Description

This class is used for managing images originated from a line scan sensor.

When capturing with a line scan sensor, HERMES grabber devices return captured lines grouped together as a "page". Page height is constant, but defined by the end user during setup time (typically, the page height is set via the vertical resolution).

Small page size (a few lines) allows fast respons time and good time stamping with the drawback that system load is significant. Limiting page rate below 1000 pages per second will assure stability. As an example, using a line scan sensor at 20000 lines/s line rate, a good page size would be 100 lines, hence limiting the page rate to 100 pages per second.

This class can be used to process line scan pages and render them. Waterfall image a typical renderring for line scan. Waterfall images can either have a top to bottom orientation or vice versa. Waterfall images does not necessary have vertical height equal to a page height. Waterfall images are handle as a FIFO queue, where lines can be either added at the front or at the back. Combine this class with HImageRotation in order to generate horizontal waterfall renderring.

The class can generate waterfall images from 2 different sources: live (capture device) or from sequence file. Generated waterfall images can be exported to movie files or individual image files. Time stamp on waterfall images are corresponding to each page time stamp.

3.45.2 Member Function Documentation

3.45.2.1 void HLineScanManager::Configure (HImage * *image,* int *lineCount*)

Configure the line scan renderring.

Parameters

in	*image*	A reference image to be used for configuration. Width and height as weel as image format.
in	*lineCount*	Number of total lines to be displayed.

3.45.2.2 void HLineScanManager::SetLineCount (int *lineCount*)

Set the number of total lines to be displayed.

Parameters

in	*lineCount*	Number of total lines to be displayed.

3.45.2.3 void HLineScanManager::SetDirection (int *order,* int *orientation,* bool *jump*)

Set the waterfall flow direction. Depends on scan direction.

Parameters

in	*order*	0 - Start with first page in queue, 1 - start with last page in queue.
in	*orientation*	0 - forward (add pages at the end of the current set), 1 - backward (add pages at the beginning of the current set).
in	*jump*	True to clear all images that are currently in queue. False otherwise.

3.45.2.4 void HLineScanManager::PushImageToQueue (HImage * *image*)

Add image to the line scan manager FIFO queue of images.

Parameters

in	*image*	Image to be added to the queue.

3.45.2.5 bool HLineScanManager::GetLiveImage (HImage * *lineScanImage*)

Build a resulting image from a live stream source.

Returns

True if the function is successful or false if the function failed.

Parameters

out	*lineScanImage*	Resulting image.

3.45.2.6 bool HLineScanManager::GetImageFromSeq (HImage * *lineScanImage,* HSequence * *sequence,* int *indexPosition*)

Build a resulting image from a previously recorded sequence file.

Returns

True if the function is successful or false if the function failed.

Parameters

out	*lineScanImage*	Resulting image.
in	*sequence*	Sequence file handler to be used as a source.
in	*indexPosition*	Index position in the sequence file.

3.45.2.7 bool HLineScanManager::ConfigureImageExport (int *digits,* eHJpegQuality *jpegQuality,* bool *multiTIFF*)

Configure the line scan manager for exporting a waterfall renderred image.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*digits*	Padding for file names (i.e. for image0001 use value 4).
in	*jpegQuality*	JPEG Quality (useful only when exporting to JPEG).
in	*multiTIFF*	True if exporting to Multi TIFF, otherwise false.

3.45.2.8 bool HLineScanManager::ConfigureAVIExport (int *videoCodec,* int *audioSource,* int *audioCodec*)

Configure the line scan manager for exporting a waterfall renderred AVI.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*videoCodec*	Index of the video codec used.
in	*audioSource*	Index of the audio source used.
in	*audioCodec*	Index of the audio codec used.

3.45.2.9 bool HLineScanManager::ConfigureMOVExport (int *videoCodec,* int *audioSource,* int *videoQuality,* unsigned long *audioSampleRate,* short *audioSampleSize,* short *audioChannelCount,* unsigned long *audioCodecType*)

Configure the line scan manager for exporting a waterfall renderred MOV (32 bit only).

Returns

True if the function is successful or false if the function failed.

Parameters

in	*videoCodec*	Index of the video codec used.
in	*audioSource*	Index of the audio source used.
in	*videoQuality*	Video quality.
in	*audioSample↩ Rate*	Audio sample rate.
in	*audioSample↩ Size*	Audio sample size.
in	*audioChannel↩ Count*	Number of audio channels.
in	*audioCodecType*	Index of the audio codec used.

3.45.2.10 bool HLineScanManager::Export (HSequence * *sequence,* int *destImageHeight,* int *startFrame,* int *endFrame,* eHExportFormat *fileFormat,* LPCTSTR *destName,* int *rotationAngle* = 0, double *frameRate* = 30)

Export into a waterfall renderred sequence file.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*sequence*	Sequence to be used for exporting.
in	*destImageHeight*	The height of the destination image (number of lines).
in	*startFrame*	Starting position in the sequence of the images to be exported.
in	*endFrame*	Ending position in the sequence of the images to be exported.
in	*fileFormat*	The destination format for the export.
in	*destName*	The destination name for the export.
in	*rotationAngle*	Rotation angle for the exported line scan image (0, 90, 180, -90).
in	*frameRate*	Force a frame rate for SEQ, AVI or MOV export.

3.45.2.11 int HLineScanManager::GetCurrentExportedFrame ( )

Get the image currently being exported.

Returns

The playback position of the image currently being exported.

3.45.2.12 bool HLineScanManager::GetTimeStampImages (HImage * *top,* HImage * *center,* HImage * *bottom*)

Retreive reference images for displaying timestamps on the resulting line scan image.

This is using the time stamp from the souce page image

Returns

True if the function is successful or false if the function failed.

Parameters

out	*top*	reference image from the top of the waterfall.
out	*center*	reference image from the center of the waterfall.
out	*bottom*	reference image from the bottom of the waterfall.

3.45.2.13 void HLineScanManager::Configure (HImage * *image*, int *lineCount*)

Configure the line scan renderring.

Parameters

in	*image*	A reference image to be used for configuration. Width and height as weel as image format.
in	*lineCount*	Number of total lines to be displayed.

3.45.2.14 void HLineScanManager::SetLineCount (int *lineCount*)

Set the number of total lines to be displayed.

Parameters

in	*lineCount*	Number of total lines to be displayed.

3.45.2.15 void HLineScanManager::SetDirection (int *order*, int *orientation*, bool *jump*)

Set the waterfall flow direction. Depends on scan direction.

Parameters

in	*order*	0 - Start with first page in queue, 1 - start with last page in queue.
in	*orientation*	0 - forward (add pages at the end of the current set), 1 - backward (add pages at the beginning of the current set).
in	*jump*	True to clear all images that are currently in queue. False otherwise.

3.45.2.16 void HLineScanManager::PushImageToQueue (HImage * *image*)

Add image to the line scan manager FIFO queue of images.

Parameters

in	*image*	Image to be added to the queue.

3.45.2.17 bool HLineScanManager::GetLiveImage (HImage * *lineScanImage*)

Build a resulting image from a live stream source.

Returns

True if the function is successful or false if the function failed.

Parameters

out	*lineScanImage*	Resulting image.

3.45.2.18 bool HLineScanManager::GetImageFromSeq (HImage * *lineScanImage,* HSequence * *sequence,* int *indexPosition*)

Build a resulting image from a previously recorded sequence file.

Returns

True if the function is successful or false if the function failed.

Parameters

out	*lineScanImage*	Resulting image.
in	*sequence*	Sequence file handler to be used as a source.
in	*indexPosition*	Index position in the sequence file.

3.45.2.19 bool HLineScanManager::ConfigureImageExport (int *digits,* eHJpegQuality *jpegQuality,* bool *multiTIFF*)

Configure the line scan manager for exporting a waterfall renderred image.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*digits*	Padding for file names (i.e. for image0001 use value 4).
in	*jpegQuality*	JPEG Quality (useful only when exporting to JPEG).
in	*multiTIFF*	True if exporting to Multi TIFF, otherwise false.

3.45.2.20 bool HLineScanManager::ConfigureAVIExport (int *videoCodec,* int *audioSource,* int *audioCodec*)

Configure the line scan manager for exporting a waterfall renderred AVI.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*videoCodec*	Index of the video codec used.
in	*audioSource*	Index of the audio source used.
in	*audioCodec*	Index of the audio codec used.

3.45.2.21 bool HLineScanManager::ConfigureMOVExport (int *videoCodec,* int *audioSource,* int *videoQuality,* unsigned long *audioSampleRate,* short *audioSampleSize,* short *audioChannelCount,* unsigned long *audioCodecType*)

Configure the line scan manager for exporting a waterfall renderred MOV (32 bit only).

Returns

True if the function is successful or false if the function failed.

Parameters

in	*videoCodec*	Index of the video codec used.
in	*audioSource*	Index of the audio source used.
in	*videoQuality*	Video quality.
in	*audioSample↩ Rate*	Audio sample rate.
in	*audioSample↩ Size*	Audio sample size.
in	*audioChannel↩ Count*	Number of audio channels.
in	*audioCodecType*	Index of the audio codec used.

3.45.2.22 bool HLineScanManager::Export (HSequence * *sequence,* int *destImageHeight,* int *startFrame,* int *endFrame,* eHExportFormat *fileFormat,* LPCTSTR *destName,* int *rotationAngle* = 0, double *frameRate* = 30)

Export into a waterfall renderred sequence file.

Returns

True if the function is successful or false if the function failed.

Parameters

in	*sequence*	Sequence to be used for exporting.
in	*destImageHeight*	The height of the destination image (number of lines).
in	*startFrame*	Starting position in the sequence of the images to be exported.
in	*endFrame*	Ending position in the sequence of the images to be exported.
in	*fileFormat*	The destination format for the export.
in	*destName*	The destination name for the export.
in	*rotationAngle*	Rotation angle for the exported line scan image (0, 90, 180, -90).
in	*frameRate*	Force a frame rate for SEQ, AVI or MOV export.

3.45.2.23 int HLineScanManager::GetCurrentExportedFrame ( )

Get the image currently being exported.

Returns

The playback position of the image currently being exported.

3.45.2.24 bool HLineScanManager::GetTimeStampImages (HImage * *top,* HImage * *center,* HImage * *bottom*)

Retreive reference images for displaying timestamps on the resulting line scan image.

This is using the time stamp from the souce page image

Returns

True if the function is successful or false if the function failed.

Parameters

out	*top*	reference image from the top of the waterfall.
out	*center*	reference image from the center of the waterfall.
out	*bottom*	reference image from the bottom of the waterfall.

3.46 HLocker Class Reference

A mutex class used through the API.

`#include <HLocker.h>`

Public Member Functions

- HLocker ()

  Initialize the critical section.
- ~HLocker ()

  Release resources used by the critical section object.
- void Lock ()

  Enter the critical sction (will block if section is already owned)
- bool TryToLock ()

  Attempt to enter the critical section, but won't block if it can't.
- void Unlock ()

  Release ownership of the critical section.

3.46.1 Detailed Description

A mutex class used through the API.

3.46.2 Member Function Documentation

3.46.2.1 bool HLocker::TryToLock ( ) [inline]

Attempt to enter the critical section, but won't block if it can't.

Returns

True if it entered the critical section, false otherwise.

3.47 HMetadata Class Reference

Base class that manages a metadata information.

`#include <HMetadata.h>`

Inherited by HMetadataGenericBinary, HMetadataGenericBool, HMetadataGenericByte, HMetadataGenericDouble, HMetadataGenericInt, HMetadataGenericInt64, HMetadataGenericShort, HMetadataGenericString, HMetadataGenericUInt, HMetadataGenericUInt64, HMetadataGenericUShort, HMetadataNorPixCameraInfo, HMetadataNorPixDMIExtended, HMetadataNorPixDMIFBExtended, HMetadataNorPixGPS, HMetadataNorPixGPSExtended, HMetadataNorPixGRTHeader, HMetadataNorPixImageStatistics, HMetadataNorPixInputLineState, HMetadataNorPixLidar, HMetadataNorPixLTC, and HMetadataNorPixTime64.

Public Member Functions

- HMetadata ()

  Create an empty metadata.
- HMetadata (unsigned int uID, unsigned int dataSizeBytes, BYTE ∗data, wchar_t ∗metadataString)

  Create a metadata from scratch.
- HMetadata (const HMetadata &src)

  Create an exact copy of an existing metadata.
- void operator= (const HMetadata &src)

  Create an exact copy of an existing metadata.
- HMetadata (BYTE ∗src)

  Create a metadata from a byte array.
- void BuildFrom (const HMetadata &src)

  Copy metadata from another metadata.
- unsigned int GetUID () const

  Get the metadata unique identifier.
- unsigned int GetDataSizeBytes () const

  Get the size of the data.
- BYTE ∗ GetData () const

  Get a pointer to the data.
- void SetMetadataString (wchar_t ∗src)

  Set the string representation for this metadata.
- wchar_t ∗ GetMetadataString ()

  Get a string representation of the metadata.
- int GetPacketSize ()

  Get the size needed to store the whole metadata structure.
- void WritePacketTo (BYTE ∗dest)

  Dump the whole metadata structure to a byte array.

Protected Attributes

- unsigned int UID

  Metadata Unique Identifer.
- unsigned int DataSizeBytes

  Metadata size.
- BYTE ∗ Data

  The metadata itself.
- wchar_t ∗ MetadataString

  String representation of the metadata (only used at runtime)

3.47.1 Detailed Description

Base class that manages a metadata information.

It has an unique identifer that can be used to know which HMetadata derived class can be use to decipher the metadata content. The HMtadata class itself only hold the metadata and its size, along with a string representation of the metadata (if available).

3.47.2 Constructor & Destructor Documentation

3.47.2.1 HMetadata::HMetadata (unsigned int *uID*, unsigned int *dataSizeBytes*, BYTE ∗ *data*, wchar_t ∗ *metadataString*) [inline]

Create a metadata from scratch.

Parameters

in	*uID*	The unique ID for this metadata type.
in	*dataSizeBytes*	The size in bytes of the "data" param.
in	*data*	The metadata
in	*metadataString*	A string representation of the metadata or NULL.

3.47.2.2 HMetadata::HMetadata (const HMetadata & *src*) [inline]

Create an exact copy of an existing metadata.

Parameters

in	*src*	The source metadata

3.47.2.3 HMetadata::HMetadata (BYTE * *src*) [inline]

Create a metadata from a byte array.

See also

WritePacketTo

Parameters

in	*src*	The byte array

3.47.3 Member Function Documentation

3.47.3.1 void HMetadata::operator= (const HMetadata & *src*) [inline]

Create an exact copy of an existing metadata.

Parameters

in	*src*	The source metadata

3.47.3.2 void HMetadata::BuildFrom (const HMetadata & *src*) [inline]

Copy metadata from another metadata.

Parameters

in	*src*	The source metadata

3.47.3.3 unsigned int HMetadata::GetUID ( ) const [inline]

Get the metadata unique identifier.

Returns

The unique ID.

3.47.3.4 unsigned int HMetadata::GetDataSizeBytes ( ) const [inline]

Get the size of the data.

Returns

The data size in bytes.

3.47.3.5 BYTE* HMetadata::GetData ( ) const [inline]

Get a pointer to the data.

Returns

A pointer to the data.

3.47.3.6 void HMetadata::SetMetadataString (wchar_t * *src*) [inline]

Set the string representation for this metadata.

Parameters

in	*src*	A string representation of the metadata or NULL.

3.47.3.7 wchar_t* HMetadata::GetMetadataString ( ) [inline]

Get a string representation of the metadata.

Returns

The string representation of the metadata.

3.47.3.8 int HMetadata::GetPacketSize ( ) [inline]

Get the size needed to store the whole metadata structure.

This is 4 bytes for the UID + 4 for the data size + the data itself.

Returns

The number of bytes needed to store the whole metadata structure.

3.47.3.9 void HMetadata::WritePacketTo (BYTE * *dest*) [inline]

Dump the whole metadata structure to a byte array.

Call GetPacketSize to know how many bytes are needed. The content of the metadata is stored in this order : [UID][SIZE][DATA]

See also

GetPacketSize

Parameters

in,out	*dest*	The byte array where to save the metadata

3.48 HMetadataGenericBinary Class Reference

Binary data of variable length.

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataGenericBinary (BYTE *data, unsigned int dataSizeBytes)

  Create a generic array of bytes metadata.
- HMetadataGenericBinary (unsigned int customID, BYTE *data, unsigned int dataSizeBytes)

  Create a custom array of bytes metadata.
- HMetadataGenericBinary (const HMetadata &src)

  Create an array of bytes from another a source metadata.

Additional Inherited Members

3.48.1 Detailed Description

Binary data of variable length.

3.48.2 Constructor & Destructor Documentation

3.48.2.1 HMetadataGenericBinary::HMetadataGenericBinary (BYTE * *data*, unsigned int *dataSizeBytes*) [inline]

Create a generic array of bytes metadata.

Parameters

in	*data*	The array of bytes.
in	*dataSizeBytes*	The length of the array (in bytes)

3.48.2.2 HMetadataGenericBinary::HMetadataGenericBinary (unsigned int *customID*, BYTE * *data*, unsigned int *dataSizeBytes*) [inline]

Create a custom array of bytes metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_USER_LAST]
in	*data*	The array of bytes.
in	*dataSizeBytes*	The length of the array (in bytes)

3.48.2.3 HMetadataGenericBinary::HMetadataGenericBinary (const HMetadata & *src*) [inline]

Create an array of bytes from another a source metadata.

Parameters

in	*src*	The source metadata

3.49 HMetadataGenericBool Class Reference

1 byte "bool" value

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataGenericBool (bool value)

  Create a generic boolean metadata.
- HMetadataGenericBool (UINT customID, bool value)

  Create a custom boolean metadata.
- HMetadataGenericBool (const HMetadata &src)

  Create a boolean metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata ∗src, wchar_t ∗format=NULL)

  Rebuild the string representation of the boolean metadata.

Additional Inherited Members

3.49.1 Detailed Description

1 byte "bool" value

3.49.2 Constructor & Destructor Documentation

3.49.2.1 HMetadataGenericBool::HMetadataGenericBool (bool *value*) [inline]

Create a generic boolean metadata.

Parameters

in	*value*	The initial bool value.

3.49.2.2 HMetadataGenericBool::HMetadataGenericBool (UINT *customID,* bool *value*) [inline]

Create a custom boolean metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]

in	*value*	The initial bool value.

3.49.2.3 HMetadataGenericBool::HMetadataGenericBool (const HMetadata & *src*) [inline]

Create a boolean metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.49.3 Member Function Documentation

3.49.3.1 static void HMetadataGenericBool::BuildMetadataString (HMetadata * *src*, wchar_t * *format* = NULL) [inline], [static]

Rebuild the string representation of the boolean metadata.

Parameters

in	*src*	The source boolean metadata.
in	*format*	If not NULL, will replace the default "true" and "false" by a custom string. For example : "Device enabled;Device disabled".

3.50 HMetadataGenericByte Class Reference

1 byte value

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataGenericByte (BYTE value)

  Create a generic 1 byte metadata.
- HMetadataGenericByte (unsigned int customID, BYTE value)

  Create a custom 1 byte metadata.
- HMetadataGenericByte (const HMetadata &src)

  Create a 1 byte metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the 1 byte metadata.

Additional Inherited Members

3.50.1 Detailed Description

1 byte value

3.50.2 Constructor & Destructor Documentation

3.50.2.1 HMetadataGenericByte::HMetadataGenericByte (BYTE *value*) [inline]

Create a generic 1 byte metadata.

Parameters

in	*value*	The initial 1 byte value.

3.50.2.2 HMetadataGenericByte::HMetadataGenericByte (unsigned int *customID*, BYTE *value*) [inline]

Create a custom 1 byte metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]
in	*value*	The initial 1 byte value.

3.50.2.3 HMetadataGenericByte::HMetadataGenericByte (const HMetadata & *src*) [inline]

Create a 1 byte metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.50.3 Member Function Documentation

3.50.3.1 static void HMetadataGenericByte::BuildMetadataString (HMetadata * *src*, wchar_t * *format* = NULL) [inline], [static]

Rebuild the string representation of the 1 byte metadata.

Parameters

in	*src*	The source 1 byte metadata.
in	*format*	If not NULL, will replace the default "%d" by a custom string in printf format. For example : "Length : %d".

3.51 HMetadataGenericDouble Class Reference

8 bytes "double" value

```
#include <NpxMetadata.h>
```

Inherits HMetadata.

Public Member Functions

- HMetadataGenericDouble (double value)

  Create a generic double metadata.
- HMetadataGenericDouble (unsigned int customID, double value)

  Create a custom double metadata.

- HMetadataGenericDouble (const HMetadata &src)

  Create a double metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the double metadata.

Additional Inherited Members

3.51.1 Detailed Description

8 bytes "double" value

3.51.2 Constructor & Destructor Documentation

3.51.2.1 HMetadataGenericDouble::HMetadataGenericDouble (double *value*) [inline]

Create a generic double metadata.

Parameters

in	*value*	The initial double value.

3.51.2.2 HMetadataGenericDouble::HMetadataGenericDouble (unsigned int *customID*, double *value*) [inline]

Create a custom double metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]
in	*value*	The initial double value.

3.51.2.3 HMetadataGenericDouble::HMetadataGenericDouble (const HMetadata & *src*) [inline]

Create a double metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.51.3 Member Function Documentation

3.51.3.1 static void HMetadataGenericDouble::BuildMetadataString (HMetadata * *src*, wchar_t * *format* = NULL) [inline], [static]

Rebuild the string representation of the double metadata.

Parameters

in	*src*	The source double metadata.
in	*format*	If not NULL, will replace the default "%f" by a custom string in printf format. For example : "Length : %.02f".

3.52 HMetadataGenericInt Class Reference

4 bytes "int" value

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataGenericInt (int value)

  Create a generic int metadata.
- HMetadataGenericInt (unsigned int customID, int value)

  Create a custom int metadata.
- HMetadataGenericInt (const HMetadata &src)

  Create a int metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the int metadata.

Additional Inherited Members

3.52.1 Detailed Description

4 bytes "int" value

3.52.2 Constructor & Destructor Documentation

3.52.2.1 HMetadataGenericInt::HMetadataGenericInt (int *value*) [inline]

Create a generic int metadata.

Parameters

in	*value*	The initial int value.

3.52.2.2 HMetadataGenericInt::HMetadataGenericInt (unsigned int *customID*, int *value*) [inline]

Create a custom int metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]
in	*value*	The initial int value.

3.52.2.3 HMetadataGenericInt::HMetadataGenericInt (const HMetadata & *src*) [inline]

Create a int metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.52.3 Member Function Documentation

3.52.3.1 static void HMetadataGenericInt::BuildMetadataString (HMetadata * *src*, wchar_t * *format* = NULL) [inline], [static]

Rebuild the string representation of the int metadata.

Parameters

in	*src*	The source int metadata.
in	*format*	If not NULL, will replace the default "%d" by a custom string in printf format. For example : "Length : %d".

3.53 HMetadataGenericInt64 Class Reference

8 bytes "int64" value

```
#include <NpxMetadata.h>
```

Inherits HMetadata.

Public Member Functions

- HMetadataGenericInt64 (__int64 value)

  Create a generic int64 metadata.
- HMetadataGenericInt64 (unsigned int customID, __int64 value)

  Create a custom int64 metadata.
- HMetadataGenericInt64 (const HMetadata &src)

  Create a int64 metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the int64 metadata.

Additional Inherited Members

3.53.1 Detailed Description

8 bytes "int64" value

3.53.2 Constructor & Destructor Documentation

3.53.2.1 HMetadataGenericInt64::HMetadataGenericInt64 (__int64 *value*) [inline]

Create a generic int64 metadata.

Parameters

in	*value*	The initial int64 value.

3.53.2.2 HMetadataGenericInt64::HMetadataGenericInt64 (unsigned int *customID*, __int64 *value*) [inline]

Create a custom int64 metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]
in	*value*	The initial int64 value.

3.53.2.3 HMetadataGenericInt64::HMetadataGenericInt64 (const **HMetadata** & *src*) [inline]

Create a int64 metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.53.3 Member Function Documentation

3.53.3.1 static void HMetadataGenericInt64::BuildMetadataString (**HMetadata** * *src*, wchar_t * *format* = NULL) [inline], [static]

Rebuild the string representation of the int64 metadata.

Parameters

in	*src*	The source int metadata.
in	*format*	If not NULL, will replace the default "%I64d" by a custom string in printf format. For example : "Length : %I64d".

3.54 HMetadataGenericShort Class Reference

2 bytes "short" value HMetadataGenericBinary - binary data (variable length)

```
#include <NpxMetadata.h>
```

Inherits HMetadata.

Public Member Functions

- HMetadataGenericShort (short value)

  Create a generic short metadata.
- HMetadataGenericShort (unsigned int customID, short value)

  Create a custom short metadata.

- HMetadataGenericShort (const HMetadata &src)

  Create a short metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the short metadata.

Additional Inherited Members

3.54.1 Detailed Description

2 bytes "short" value HMetadataGenericBinary - binary data (variable length)

3.54.2 Constructor & Destructor Documentation

3.54.2.1 HMetadataGenericShort::HMetadataGenericShort (short *value*) [inline]

Create a generic short metadata.

Parameters

in	*value*	The initial short value.

3.54.2.2 HMetadataGenericShort::HMetadataGenericShort (unsigned int *customID*, short *value*) [inline]

Create a custom short metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]
in	*value*	The initial short value.

3.54.2.3 HMetadataGenericShort::HMetadataGenericShort (const HMetadata & *src*) [inline]

Create a short metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.54.3 Member Function Documentation

3.54.3.1 static void HMetadataGenericShort::BuildMetadataString (HMetadata * *src*, wchar_t * *format* = NULL) [inline], [static]

Rebuild the string representation of the short metadata.

Parameters

in	*src*	The source short metadata.
in	*format*	If not NULL, will replace the default "%d" by a custom string in printf format. For example : "Length : %d".

3.55 HMetadataGenericString Class Reference

"wchar_t" null-terminated string (variable length)

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataGenericString (wchar_t *value)

  Create a generic string metadata.
- HMetadataGenericString (unsigned int customID, wchar_t *value)

  Create a custom string metadata.
- HMetadataGenericString (const HMetadata &src)

  Create a string metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the string metadata.

Additional Inherited Members

3.55.1 Detailed Description

"wchar_t" null-terminated string (variable length)

3.55.2 Constructor & Destructor Documentation

3.55.2.1 HMetadataGenericString::HMetadataGenericString (wchar_t * *value*) [inline]

Create a generic string metadata.

Parameters

in	*value*	The initial string value.

3.55.2.2 HMetadataGenericString::HMetadataGenericString (unsigned int *customID,* wchar_t * *value*) [inline]

Create a custom string metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_USER_LAST]
in	*value*	The initial string value.

3.55.2.3 HMetadataGenericString::HMetadataGenericString (const HMetadata & *src*) [inline]

Create a string metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.55.3 Member Function Documentation

3.55.3.1 static void HMetadataGenericString::BuildMetadataString (HMetadata * *src*, wchar_t * *format* = NULL) [inline], [static]

Rebuild the string representation of the string metadata.

Parameters

in	*src*	The source string metadata.
in	*format*	If not NULL, will replace the default "%s" by a custom string in printf format. For example : "Description : %s".

3.56 HMetadataGenericUInt Class Reference

4 bytes "unsigned int" value

```
#include <NpxMetadata.h>
```

Inherits HMetadata.

Public Member Functions

- HMetadataGenericUInt (unsigned int value)

  Create a generic unsigned int metadata.
- HMetadataGenericUInt (unsigned int customID, unsigned int value)

  Create a custom unsigned int metadata.
- HMetadataGenericUInt (const HMetadata &src)

  Create a unsigned int metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the unsigned int metadata.

Additional Inherited Members

3.56.1 Detailed Description

4 bytes "unsigned int" value

3.56.2 Constructor & Destructor Documentation

3.56.2.1 HMetadataGenericUInt::HMetadataGenericUInt (unsigned int *value*) [inline]

Create a generic unsigned int metadata.

Parameters

in	*value*	The initial unsigned int value.

3.56.2.2 HMetadataGenericUInt::HMetadataGenericUInt (unsigned int *customID,* unsigned int *value*) [inline]

Create a custom unsigned int metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]
in	*value*	The initial unsigned int.

3.56.2.3 HMetadataGenericUInt::HMetadataGenericUInt (const HMetadata & *src*) [inline]

Create a unsigned int metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.56.3 Member Function Documentation

3.56.3.1 static void HMetadataGenericUInt::BuildMetadataString (HMetadata * *src,* wchar_t * *format* = NULL) [inline],[static]

Rebuild the string representation of the unsigned int metadata.

Parameters

in	*src*	The source unsigned int metadata.
in	*format*	If not NULL, will replace the default "%u" by a custom string in printf format. For example : "Length : %u".

3.57 HMetadataGenericUInt64 Class Reference

8 bytes "unsigned int64" value

```
#include <NpxMetadata.h>
```

Inherits HMetadata.

Public Member Functions

- HMetadataGenericUInt64 (unsigned __int64 value)

  Create a generic unsigned int64 metadata.
- HMetadataGenericUInt64 (unsigned int customID, unsigned __int64 value)

  Create a custom unsigned int64 metadata.

- HMetadataGenericUInt64 (const HMetadata &src)

  Create a unsigned int64 metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the unsigned int64 metadata.

Additional Inherited Members

3.57.1 Detailed Description

8 bytes "unsigned int64" value

3.57.2 Constructor & Destructor Documentation

3.57.2.1 HMetadataGenericUInt64::HMetadataGenericUInt64 (unsigned __int64 *value*) [inline]

Create a generic unsigned int64 metadata.

Parameters

in	*value*	The initial unsigned int64 value.

3.57.2.2 HMetadataGenericUInt64::HMetadataGenericUInt64 (unsigned int *customID*, unsigned __int64 *value*) [inline]

Create a custom unsigned int64 metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]
in	*value*	The initial unsigned int64.

3.57.2.3 HMetadataGenericUInt64::HMetadataGenericUInt64 (const HMetadata & *src*) [inline]

Create a unsigned int64 metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.57.3 Member Function Documentation

3.57.3.1 static void HMetadataGenericUInt64::BuildMetadataString (HMetadata * *src*, wchar_t * *format* = NULL) [inline],[static]

Rebuild the string representation of the unsigned int64 metadata.

Parameters

in	*src*	The source unsigned int64 metadata.
in	*format*	If not NULL, will replace the default "%I64u" by a custom string in printf format. For example : "Length : %I64u".

3.58 HMetadataGenericUShort Class Reference

2 bytes "unsigned short" value

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataGenericUShort (unsigned short value)

  Create a generic unsigned short metadata.
- HMetadataGenericUShort (unsigned int customID, unsigned short value)

  Create a custom unsigned short metadata.
- HMetadataGenericUShort (const HMetadata &src)

  Create a unsigned short metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src, wchar_t *format=NULL)

  Rebuild the string representation of the unsigned short metadata.

Additional Inherited Members

3.58.1 Detailed Description

2 bytes "unsigned short" value

3.58.2 Constructor & Destructor Documentation

3.58.2.1 HMetadataGenericUShort::HMetadataGenericUShort (unsigned short *value*) [inline]

Create a generic unsigned short metadata.

Parameters

in	*value*	The initial unsigned short value.

3.58.2.2 HMetadataGenericUShort::HMetadataGenericUShort (unsigned int *customID*, unsigned short *value*) [inline]

Create a custom unsigned short metadata.

Parameters

in	*customID*	A custom ID between [UID_METADATA_USER_FIRST, UID_METADATA_↩ USER_LAST]
in	*value*	The initial unsigned short value.

3.58.2.3 HMetadataGenericUShort::HMetadataGenericUShort (const HMetadata & *src*) [inline]

Create a unsigned short metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.58.3 Member Function Documentation

3.58.3.1 static void HMetadataGenericUShort::BuildMetadataString (HMetadata * *src*, wchar_t * *format* = NULL) [inline], [static]

Rebuild the string representation of the unsigned short metadata.

Parameters

in	*src*	The source unsigned short metadata.
in	*format*	If not NULL, will replace the default "%d" by a custom string in printf format. For example : "Length : %d".

3.59 HMetadataInfo Class Reference

Manage all the information needed to handle each metadata type.

```
#include <HMetadataInfo.h>
```

Public Member Functions

- HMetadataInfo (unsigned int uID, wchar_t *name, wchar_t *description, eHMetadataFormat format, unsigned int size, wchar_t *formatString)

  Create a metadata info from scratch.
- HMetadataInfo (const HMetadataInfo &info)

  Copy from another metadata info.
- HMetadataInfo (BYTE *src)

  Build a metadata info from a byte array.
- int GetPacketSize ()

  Get the size needed to store the whole metadata info structure.
- void WritePacketTo (BYTE *dest)

  Dump the whole metadata info structure to a byte array.

Public Attributes

- unsigned int UID

  Metadata unique identifer.
- wchar_t * Name

Metadata name.

- wchar_t * Description

  Metadata full name.

- eHMetadataFormat Format

  Metadata format.

- unsigned int MetadataSize

  Metadata size in bytes.

- wchar_t * FormatString

  Metadata format string (printf format).

3.59.1 Detailed Description

Manage all the information needed to handle each metadata type.

3.59.2 Constructor & Destructor Documentation

3.59.2.1 HMetadataInfo::HMetadataInfo (unsigned int *uID*, wchar_t * *name*, wchar_t * *description*, eHMetadataFormat *format*, unsigned int *size*, wchar_t * *formatString*) [inline]

Create a metadata info from scratch.

Parameters

in	*uID*	The metadata unique identifer. Check MetadataID.h for existing IDs.
in	*name*	The metadata name.
in	*description*	The metadata description.
in	*format*	The metadata format.
in	*size*	The metadata size in bytes.
in	*formatString*	The string used to format the metadata value (in "printf" format). For example, "Lenght : %d meters".

3.59.2.2 HMetadataInfo::HMetadataInfo (const HMetadataInfo & *info*) [inline]

Copy from another metadata info.

Parameters

in	*info*	The source metadata

3.59.3 Member Function Documentation

3.59.3.1 int HMetadataInfo::GetPacketSize ( ) [inline]

Get the size needed to store the whole metadata info structure.

Returns

The number of bytes needed to store the whole metadata info structure.

3.59.3.2 void HMetadataInfo::WritePacketTo (BYTE * *dest*) [inline]

Dump the whole metadata info structure to a byte array.

Call GetPacketSize to know how many bytes are needed.

See also

GetPacketSize

Parameters

in,out	*dest*	The byte array where to save the metadata info

3.60 HMetadataManager Class Reference

Manage the metadata information stored in the registry.

`#include <HMetadataManager.h>`

Public Member Functions

- bool RegisterMetadataType (HMetadataInfo *info)

  Register metadata information for a 3rd party metadata type.
- bool RegisterGenericTypes ()

  Register all the generic metadata types informations.
- bool RegisterNorPixTypes ()

  Register all the NorPix metadata types informations.
- int GetMetadataTypeCount ()

  Retrieve the total count of metadata types registered on the system.
- HMetadataInfo * GetMetadataTypeByIndex (unsigned int index)

  Retrieve the metadata information from the registry, for the metadata type occupying this index.
- HMetadataInfo * GetMetadataTypeByID (unsigned int id)

  Retrieve the metadata information from the registry, for the metadata type having this unique identifier.
- void RebuildMetadataString (HMetadata *metadata)

  Rebuild the string representation of the metadata. This can be done for all generic/NorPix types. For 3rd party types, the HMetadataInfo's FormatString will be used (if available) to rebuild the string.

3.60.1 Detailed Description

Manage the metadata information stored in the registry.

3.60.2 Member Function Documentation

3.60.2.1 bool HMetadataManager::RegisterMetadataType (HMetadataInfo * *info*)

Register metadata information for a 3rd party metadata type.

Returns

True if the value was successfully written to the registry. False otherwise.

Parameters

in	*info*	The metadata information.

3.60.2.2 bool HMetadataManager::RegisterGenericTypes ( )

Register all the generic metadata types informations.

Returns

True if the registration was successful. False otherwise.

3.60.2.3 bool HMetadataManager::RegisterNorPixTypes ( )

Register all the NorPix metadata types informations.

Returns

True if the registration was successful. False otherwise.

3.60.2.4 int HMetadataManager::GetMetadataTypeCount ( )

Retrieve the total count of metadata types registered on the system.

This include any registered generic types, NorPix types and 3rd party types.

Returns

The total count of metadata types registered on the system.

3.60.2.5 HMetadataInfo* HMetadataManager::GetMetadataTypeByIndex (unsigned int *index*)

Retrieve the metadata information from the registry, for the metadata type occupying this index.

Returns

A pointer to the metadata info or NULL if the index is out of bounds.

Parameters

in	*index*	The index, must be between [0, GetMetadataTypeCount()-1]

3.60.2.6 HMetadataInfo* HMetadataManager::GetMetadataTypeByID (unsigned int *id*)

Retrieve the metadata information from the registry, for the metadata type having this unique identifier.

Returns

A pointer to the metadata info or NULL if the unique ID is not registered on the system.

Parameters

in	*id*	The unique identifier of the metadata type.

3.60.2.7 void HMetadataManager::RebuildMetadataString (HMetadata * *metadata*)

Rebuild the string representation of the metadata. This can be done for all generic/NorPix types. For 3rd party types, the HMetadataInfo's FormatString will be used (if available) to rebuild the string.

Parameters

in,out	*metadata*	The metadata structure.

3.61 HMetadataNorPixCameraInfo Class Reference

Camera info.

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixCameraInfo (double aperture, short focalLength, double exposure, double gain, double gamma)

  Create a camera info metadata.
- HMetadataNorPixCameraInfo (const HMetadata &src)

  Create a camera info metadata from another source metadata.
- double GetAperture ()

  Get the lens aperture.
- short GetFocalLength ()

  Get the lens focal length.
- double GetExposure ()

  Get the camera exposure.
- double GetGain ()

  Get the camera gain.
- double GetGamma ()

  Get the camera gamma.
- void SetAperture (double aperture)

  Set the lens aperture value.
- void SetFocalLength (short focalLength)

  Set the lens focal length.
- void SetExposure (double exposure)

  Set the camera exposure.
- void SetGain (double gain)

  Set the camera gain.
- void SetGamma (double gamma)

  Set the camera gamma.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the camera info metadata.

Additional Inherited Members

3.61.1 Detailed Description

Camera info.

3.61.2 Constructor & Destructor Documentation

3.61.2.1 HMetadataNorPixCameraInfo::HMetadataNorPixCameraInfo (double *aperture,* short *focalLength,* double *exposure,* double *gain,* double *gamma*) [inline]

Create a camera info metadata.

Parameters

in	*aperture*	Lens aperture
in	*focalLength*	Lens focal length
in	*exposure*	Camera exposure
in	*gain*	Camera gain
in	*gamma*	Camera gamma

3.61.2.2 HMetadataNorPixCameraInfo::HMetadataNorPixCameraInfo (const **HMetadata** & *src*) [inline]

Create a camera info metadata from another source metadata.

Parameters

in	*src*	The source metadata

3.61.3 Member Function Documentation

3.61.3.1 static void HMetadataNorPixCameraInfo::BuildMetadataString (**HMetadata** * *src*) [inline], [static]

Rebuild the string representation of the camera info metadata.

<[in] Lens aperture

<[in] Lens focal length

<[in] Camera exposure

<[in] Camera gain

<[in] Camera gamma

Parameters

in	*src*	The source metadata

3.61.3.2 double HMetadataNorPixCameraInfo::GetAperture ( ) [inline]

Get the lens aperture.

Returns

the lens aperture value

3.61.3.3 short HMetadataNorPixCameraInfo::GetFocalLength ( ) [inline]

Get the lens focal length.

Returns

the lens focal length value

3.61.3.4 double HMetadataNorPixCameraInfo::GetExposure ( ) [inline]

Get the camera exposure.

Returns

the camera exposure value

3.61.3.5 double HMetadataNorPixCameraInfo::GetGain ( ) [inline]

Get the camera gain.

Returns

the camera gain value

3.61.3.6 double HMetadataNorPixCameraInfo::GetGamma ( ) [inline]

Get the camera gamma.

Returns

the camera gamma value

3.61.3.7 void HMetadataNorPixCameraInfo::SetAperture (double *aperture*) [inline]

Set the lens aperture value.

Parameters

in	*aperture*	Lens aperture value

3.61.3.8 void HMetadataNorPixCameraInfo::SetFocalLength (short *focalLength*) [inline]

Set the lens focal length.

Parameters

in	*focalLength*	Lens focal length value

3.61.3.9 void HMetadataNorPixCameraInfo::SetExposure (double *exposure*) [inline]

Set the camera exposure.

Parameters

in	*exposure*	Camera exposure value

3.61.3.10 void HMetadataNorPixCameraInfo::SetGain (double *gain*) [inline]

Set the camera gain.

Parameters

in	*gain*	Camera gain value

3.61.3.11 void HMetadataNorPixCameraInfo::SetGamma (double *gamma*) [inline]

Set the camera gamma.

Parameters

in	*gamma*	Camera gamma value

3.62 HMetadataNorPixDMIExtended Class Reference

8 bytes "DMI front mile post" value

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixDMIExtended (double milepost)

  ...
- HMetadataNorPixDMIExtended (const HMetadata &src)

  Create a DMI mile post Header metadata from another source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the DMI mile post Header metadata.

Public Attributes

- double Milepost

  ...

Additional Inherited Members

3.62.1 Detailed Description

8 bytes "DMI front mile post" value

3.62.2 Constructor & Destructor Documentation

3.62.2.1 HMetadataNorPixDMIExtended::HMetadataNorPixDMIExtended (double *milepost*) [inline]

...

Parameters

in	*milepost*	Mile post infomation

3.62.2.2 HMetadataNorPixDMIExtended::HMetadataNorPixDMIExtended (const HMetadata & *src*) [inline]

Create a DMI mile post Header metadata from another source metadata.

Parameters

src	...

3.62.3 Member Function Documentation

3.62.3.1 static void HMetadataNorPixDMIExtended::BuildMetadataString (HMetadata * *src*) [inline], [static]

Rebuild the string representation of the DMI mile post Header metadata.

Parameters

src	...

3.63 HMetadataNorPixDMIFBExtended Class Reference

16 bytes "DMI front and back mile post" value

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixDMIFBExtended (double frontMilepost, double backMilepost)

  ...
- HMetadataNorPixDMIFBExtended (const HMetadata &src)

  Create a DMI mile post with front and back Header metadata from another source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the DMI mile post with fron and back Header metadata.

Public Attributes

- double FrontMilepost

  ...
- double BackMilepost

  ...

Additional Inherited Members

3.63.1 Detailed Description

16 bytes "DMI front and back mile post" value

3.63.2 Constructor & Destructor Documentation

3.63.2.1 HMetadataNorPixDMIFBExtended::HMetadataNorPixDMIFBExtended (double *frontMilepost*, double *backMilepost*) [inline]

...

Parameters

in	*frontMilepost*	front mile post infomation
in	*backMilepost*	back mile post infomation

3.63.2.2 HMetadataNorPixDMIFBExtended::HMetadataNorPixDMIFBExtended (const **HMetadata** & *src*) [inline]

Create a DMI mile post with front and back Header metadata from another source metadata.

Parameters

src	...

3.63.3 Member Function Documentation

3.63.3.1 static void HMetadataNorPixDMIFBExtended::BuildMetadataString (**HMetadata** * *src*) [inline], [static]

Rebuild the string representation of the DMI mile post with fron and back Header metadata.

Parameters

src	...

3.64 HMetadataNorPixGPS Class Reference

GPS data (latitude, longitude)

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixGPS (double latitude, double longitude)

  Create a GPS metadata.
- HMetadataNorPixGPS (const HMetadata &src)

  Create a GPS metadata from another source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the GPS metadata.

Additional Inherited Members

3.64.1 Detailed Description

GPS data (latitude, longitude)

3.64.2 Constructor & Destructor Documentation

3.64.2.1 HMetadataNorPixGPS::HMetadataNorPixGPS (double *latitude*, double *longitude*) [inline]

Create a GPS metadata.

Parameters

in	*latitude*	The initial latitude value.
in	*longitude*	The initial longitude value.

3.64.2.2 HMetadataNorPixGPS::HMetadataNorPixGPS (const HMetadata & *src*) [inline]

Create a GPS metadata from another source metadata.

Parameters

in	*src*	The source metadata

3.64.3 Member Function Documentation

3.64.3.1 static void HMetadataNorPixGPS::BuildMetadataString (HMetadata * *src*) [inline], [static]

Rebuild the string representation of the GPS metadata.

Parameters

in	*src*	The source GPS metadata.

3.65 HMetadataNorPixGPSExtended Class Reference

GPS extended data (latitude, longitude, altitude, speed, bearing, time)

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixGPSExtended (double latitude, double longitude, double altitude, double speed, double bearing, long time)

  Create a GPS extended metadata.

- HMetadataNorPixGPSExtended (double latitude, double longitude, double altitude, double speed, double bearing, long time, short timeMS)

  Create a GPS extended metadata.
- HMetadataNorPixGPSExtended (double latitude, double longitude, double altitude, double speed, double bearing, long time, short timeMS, short timeUS)

  Create a GPS extended metadata.
- HMetadataNorPixGPSExtended (double latitude, double longitude, double altitude, double speed, double bearing, double pitch, double roll, long time, short timeMS, short timeUS)

  Create a GPS extended metadata.
- HMetadataNorPixGPSExtended (const HMetadata &src)

  Create a GPS extended metadata from another source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the GPS extended metadata.

Public Attributes

- short SubVersion

  GPS data structure version.
- double Latitude

  Latitude.
- double Longitude

  Longitude.
- double Altitude

  Altitude in meters.
- double Speed

  Speed meters per second.
- double Bearing

  Bearing.
- double Pitch

  Pitch.
- double Roll

  Roll.
- __time32_t Time

  Time.
- short TimeMS

  Millisecond.
- short TimeUS

  Microsecond.

Additional Inherited Members

3.65.1 Detailed Description

GPS extended data (latitude, longitude, altitude, speed, bearing, time)

3.65.2 Constructor & Destructor Documentation

3.65.2.1 HMetadataNorPixGPSExtended::HMetadataNorPixGPSExtended (double *latitude,* double *longitude,* double *altitude,* double *speed,* double *bearing,* long *time*) [inline]

Create a GPS extended metadata.

Parameters

in	*latitude*	The initial latitude value.
in	*longitude*	The initial longitude value.
in	*altitude*	The initial altitude value (in meters).
in	*speed*	The initial speed value (in meters/second).
in	*bearing*	The initial bearing value.
in	*time*	The initial time value.

3.65.2.2 HMetadataNorPixGPSExtended::HMetadataNorPixGPSExtended (double *latitude,* double *longitude,* double *altitude,* double *speed,* double *bearing,* long *time,* short *timeMS*) [inline]

Create a GPS extended metadata.

Parameters

in	*latitude*	The initial latitude value.
in	*longitude*	The initial longitude value.
in	*altitude*	The initial altitude value (in meters).
in	*speed*	The initial speed value (in meters/second).
in	*bearing*	The initial bearing value.
in	*time*	The initial time value.
in	*timeMS*	The initial millisecond value.

3.65.2.3 HMetadataNorPixGPSExtended::HMetadataNorPixGPSExtended (double *latitude,* double *longitude,* double *altitude,* double *speed,* double *bearing,* long *time,* short *timeMS,* short *timeUS*) [inline]

Create a GPS extended metadata.

Parameters

in	*latitude*	The initial latitude value.
in	*longitude*	The initial longitude value.
in	*altitude*	The initial altitude value (in meters).
in	*speed*	The initial speed value (in meters/second).
in	*bearing*	The initial bearing value.
in	*time*	The initial time value.
in	*timeMS*	The initial millisecond value.
in	*timeUS*	The initial microsecond value.

3.65.2.4 HMetadataNorPixGPSExtended::HMetadataNorPixGPSExtended (double *latitude,* double *longitude,* double *altitude,* double *speed,* double *bearing,* double *pitch,* double *roll,* long *time,* short *timeMS,* short *timeUS*) [inline]

Create a GPS extended metadata.

Parameters

in	*latitude*	The initial latitude value.
in	*longitude*	The initial longitude value.
in	*altitude*	The initial altitude value (in meters).
in	*speed*	The initial speed value (in meters/second).
in	*bearing*	The initial bearing value.

in	*pitch*	The initial pitch value.
in	*roll*	The initial roll value.
in	*time*	The initial time value.
in	*timeMS*	The initial millisecond value.
in	*timeUS*	The initial microsecond value.

3.65.2.5 HMetadataNorPixGPSExtended::HMetadataNorPixGPSExtended (const HMetadata & *src*) [inline]

Create a GPS extended metadata from another source metadata.

Parameters

in	*src*	The source metadata

3.65.3 Member Function Documentation

3.65.3.1 static void HMetadataNorPixGPSExtended::BuildMetadataString (HMetadata * *src*) [inline], [static]

Rebuild the string representation of the GPS extended metadata.

Parameters

in	*src*	The source metadata

3.66 HMetadataNorPixGRTHeader Class Reference

GRT Header.

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixGRTHeader (void *data, unsigned short dataSizeBytes)

  Create a GRT Header metadata.
- HMetadataNorPixGRTHeader (const HMetadata &src)

  Create a GRT Header metadata from another source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the GRT Header metadata.

Additional Inherited Members

3.66.1 Detailed Description

GRT Header.

3.66.2 Constructor & Destructor Documentation

3.66.2.1 HMetadataNorPixGRTHeader::HMetadataNorPixGRTHeader (void * *data*, unsigned short *dataSizeBytes*) [inline]

Create a GRT Header metadata.

Parameters

in	*data*	The GRT Header data
in	*dataSizeBytes*	The size of the GRT Header data

3.66.2.2 HMetadataNorPixGRTHeader::HMetadataNorPixGRTHeader (const HMetadata & *src*) [inline]

Create a GRT Header metadata from another source metadata.

Parameters

in	*src*	The source metadata

3.66.3 Member Function Documentation

3.66.3.1 static void HMetadataNorPixGRTHeader::BuildMetadataString (HMetadata * *src*) [inline],[static]

Rebuild the string representation of the GRT Header metadata.

Parameters

in	*src*	The source metadata

3.67 HMetadataNorPixImageStatistics Class Reference

Image Statistics.

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixImageStatistics (unsigned short channels, double minLevel[3], double maxLevel[3], double avgLevel[3], double stdDeviation[3], unsigned long roiOffsetX, unsigned long roiOffsetY, unsigned long roi↩SizeX, unsigned long roiSizeY, int reserved1=0, int reserved2=0, int reserved3=0, int reserved4=0)

  Create an image statistics metadata.
- HMetadataNorPixImageStatistics (const HMetadata &src)

  Create image statistics metadata from another source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the image statistics metadata.

Additional Inherited Members

3.67.1 Detailed Description

Image Statistics.

3.67.2 Constructor & Destructor Documentation

3.67.2.1 HMetadataNorPixImageStatistics::HMetadataNorPixImageStatistics (unsigned short *channels*, double *minLevel[3]*, double *maxLevel[3]*, double *avgLevel[3]*, double *stdDeviation[3]*, unsigned long *roiOffsetX*, unsigned long *roiOffsetY*, unsigned long *roiSizeX*, unsigned long *roiSizeY*, int *reserved1* = 0, int *reserved2* = 0, int *reserved3* = 0, int *reserved4* = 0) [inline]

Create an image statistics metadata.

Parameters

in	*channels*	The number of channels depending on the image format: monochrome = 1, RGB color = 3. YUV not supported.
in	*minLevel*	Minimum level
in	*maxLevel*	Maximum level
in	*avgLevel*	Average level
in	*stdDeviation*	Standard deviation
in	*roiOffsetX*	ROI offset X
in	*roiOffsetY*	ROI offset Y
in	*roiSizeX*	ROI size X
in	*roiSizeY*	ROI size Y
	reserved2	reserved
	reserved3	reserved
	reserved4	reservedreserved

3.67.2.2 HMetadataNorPixImageStatistics::HMetadataNorPixImageStatistics (const HMetadata & *src*) [inline]

Create image statistics metadata from another source metadata.

Parameters

in	*src*	The source metadata

3.67.3 Member Function Documentation

3.67.3.1 static void HMetadataNorPixImageStatistics::BuildMetadataString (HMetadata * *src*) [inline], [static]

Rebuild the string representation of the image statistics metadata.

The number of channels depending on the image format: monochrome = 1, RGB color = 3. YUV not supported.

Minimum level

Maximum level

Average level

Standard deviation

ROI offset X

ROI offset Y

ROI size X

ROI size Y

Parameters

in	*src*	The source metadata

3.68 HMetadataNorPixInputLineState Class Reference

Input line state.

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixInputLineState (const wchar_t *deviceName, const wchar_t *lineName, eHInputLevel level)

  Create an input line state metadata.
- HMetadataNorPixInputLineState (const HMetadata &src)

  Create an input line state metadata from another source metadata.
- void GetDeviceName (wchar_t *name)

  Get the name of the device.
- void GetLineName (wchar_t *name)

  Get the name of the line.
- eHInputLevel GetInputLevel ()

  Get the level of the input line.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the time metadata.

Additional Inherited Members

3.68.1 Detailed Description

Input line state.

3.68.2 Constructor & Destructor Documentation

3.68.2.1 HMetadataNorPixInputLineState::HMetadataNorPixInputLineState (const wchar_t * *deviceName*, const wchar_t * *lineName*, eHInputLevel *level*) [inline]

Create an input line state metadata.

Parameters

in	*deviceName*	The device Name.
in	*lineName*	The input line Name.
in	*level*	The input level (0=low, 1=high).

3.68.2.2 HMetadataNorPixInputLineState::HMetadataNorPixInputLineState (const HMetadata & *src*) [inline]

Create an input line state metadata from another source metadata.

Parameters

in	*src*	The source metadata

3.68.3 Member Function Documentation

3.68.3.1 static void HMetadataNorPixInputLineState::BuildMetadataString (HMetadata * *src*) [inline], [static]

Rebuild the string representation of the time metadata.

Parameters

in	*src*	The source timestamp64 metadata.

3.68.3.2 void HMetadataNorPixInputLineState::GetDeviceName (wchar_t * *name*) [inline]

Get the name of the device.

Parameters

in	*name*	The device name. Allocate at least 128 chars.

3.68.3.3 void HMetadataNorPixInputLineState::GetLineName (wchar_t * *name*) [inline]

Get the name of the line.

Parameters

in	*name*	The line name. Allocate at least 128 chars.

3.68.3.4 eHInputLevel HMetadataNorPixInputLineState::GetInputLevel ( ) [inline]

Get the level of the input line.

Returns

The level of the input line (level high / level low)

3.69 HMetadataNorPixLidar Class Reference

Lidar data.

```
#include <NpxMetadata.h>
```

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixLidar (short pitchAngle, short rollAngle, short yawAngle, short dataType, unsigned long scanFreq, unsigned short scanCnt, unsigned short angularStepWidth, float scaleOffset, long startAngle, unsigned short numberDataPoints, unsigned short *data16Pt)

  Create a Lidar metadata.
- HMetadataNorPixLidar (const HMetadata &src)

  Create a Lidar metadata from another source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the GPS extended metadata.

Additional Inherited Members

3.69.1 Detailed Description

Lidar data.

3.69.2 Member Function Documentation

3.69.2.1 static void HMetadataNorPixLidar::BuildMetadataString (HMetadata * *src*) [inline],[static]

Rebuild the string representation of the GPS extended metadata.

Parameters

in	*src*	The source metadata

3.70 HMetadataNorPixLTC Class Reference

LTC value HH:MM:SS:FF.

```
#include <NpxMetadata.h>
```

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixLTC (unsigned long ltc)

  Create a LTC metadata.
- HMetadataNorPixLTC (const HMetadata &src)

  Create a LTC metadata from another source metadata.
- unsigned long GetLTC ()

  Get the LTC value of the metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the LTC metadata.

Additional Inherited Members

3.70.1 Detailed Description

LTC value HH:MM:SS:FF.

3.70.2 Constructor & Destructor Documentation

3.70.2.1 HMetadataNorPixLTC::HMetadataNorPixLTC (unsigned long *ltc*) [inline]

Create a LTC metadata.

Parameters

in	*ltc*	The initial LTC value.

3.70.2.2 HMetadataNorPixLTC::HMetadataNorPixLTC (const HMetadata & *src*) [inline]

Create a LTC metadata from another source metadata.

Parameters

in	*src*	The source metadata

3.70.3 Member Function Documentation

3.70.3.1 static void HMetadataNorPixLTC::BuildMetadataString (HMetadata * *src*) [inline], [static]

Rebuild the string representation of the LTC metadata.

Parameters

in	*src*	The source LTC metadata.

3.70.3.2 unsigned long HMetadataNorPixLTC::GetLTC ( ) [inline]

Get the LTC value of the metadata.

Returns

The LTC value.

3.71 HMetadataNorPixTime64 Class Reference

Timestamp in time_t format (64-bit) along with MS and US precision.

`#include <NpxMetadata.h>`

Inherits HMetadata.

Public Member Functions

- HMetadataNorPixTime64 (__time64_t time, unsigned short ms, unsigned short us)

  Create a timestamp64 metadata.
- HMetadataNorPixTime64 (const HMetadata &src)

  Create a timestamp64 metadata from another a source metadata.

Static Public Member Functions

- static void BuildMetadataString (HMetadata *src)

  Rebuild the string representation of the time metadata.

Additional Inherited Members

3.71.1 Detailed Description

Timestamp in time_t format (64-bit) along with MS and US precision.

3.71.2 Constructor & Destructor Documentation

3.71.2.1 HMetadataNorPixTime64::HMetadataNorPixTime64 (__time64_t *time*, unsigned short *ms*, unsigned short *us*) [inline]

Create a timestamp64 metadata.

Parameters

in	*time*	The time_t time (64-bit).
in	*ms*	The milliseconds
in	*us*	The microseconds

3.71.2.2 HMetadataNorPixTime64::HMetadataNorPixTime64 (const HMetadata & *src*) [inline]

Create a timestamp64 metadata from another a source metadata.

Parameters

in	*src*	The source metadata

3.71.3 Member Function Documentation

3.71.3.1 static void HMetadataNorPixTime64::BuildMetadataString (HMetadata * *src*) [inline], [static]

Rebuild the string representation of the time metadata.

Parameters

in	*src*	The source timestamp64 metadata.

3.72 HMotionDetection Class Reference

This is an utility class that allows to detect object motion in images.

`#include <HMotionDetection.h>`

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()
- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

CONFIGURATION FUNCTIONS

- void SetMotionDetectionMode (eHMotionDetectionMode mode)

Set motion detection mode.
- eHMotionDetectionMode GetMotionDetectionMode ()

  Get Motion Detection Mode.
- void SetSobelDetectionMode (eHSobelMode mode)

  Set Sobel operating detection mode.
- eHSobelMode GetSobelDetectionMode ()

  Get Sobel detection mode.
- void SetThresholdValue (double value)

  Set detection threshold.
- double GetThresholdValue ()

  Get detection threshold.
- void SetAbsoluteThresholdValue (int value)

  Set Absolute Max detection threshold.
- int GetAbsoluteThresholdValue ()

  Get Absolute Max detection threshold.
- void SetRoi (int offsetX, int offsetY, int sizeX, int sizeY)

  Set Detection ROI.
- void GetRoi (int &offsetX, int &offsetY, int &sizeX, int &sizeY)

  Get Detection ROI.
- void SetMotionDetectionMode (eHMotionDetectionMode mode)

  Set motion detection mode.
- eHMotionDetectionMode GetMotionDetectionMode ()

  Get Motion Detection Mode.
- void SetSobelDetectionMode (eHSobelMode mode)

  Set Sobel operating detection mode.
- eHSobelMode GetSobelDetectionMode ()

  Get Sobel detection mode.
- void SetThresholdValue (double value)

  Set detection threshold.
- double GetThresholdValue ()

  Get detection threshold.
- void SetAbsoluteThresholdValue (int value)

  Set Absolute Max detection threshold.
- int GetAbsoluteThresholdValue ()

  Get Absolute Max detection threshold.
- void SetRoi (int offsetX, int offsetY, int sizeX, int sizeY)

  Set Detection ROI.
- void GetRoi (int &offsetX, int &offsetY, int &sizeX, int &sizeY)

  Get Detection ROI.

CONVERSION FUNCTIONS

- bool ConvertToGray (HImage *image, HImage *grayImage)

  Convert an image to the Gray 8 format.
- bool MeanGrayDetect (HImage *image, double &value)

  Perform a Mean Gray Detection.
- bool AbsDiffDetect (HImage *image, HImage *background, double &value, double &mean)

  Perform an Absolute Difference detection.
- bool AbsMaxDiffDetect (HImage *image, HImage *background, double &max)

  Perform a Max Absolute Difference detection.
- bool EdgeDetect (HImage *image, double &value)

  Perform an Edge Detection.
- bool EdgeDetect (HImage *image, HImage *resultImage, double &value)

  Perform an Edge Detection and count the number of pixels in the motion detected area.
- bool AdaptiveEdgeDetect (HImage *image, HImage *background, double &value)

  Perform an Adaptive Edge Detection.
- bool AdaptiveEdgeDetect (HImage *image, HImage *background, HImage *resultImage, double &value)

  Perform an Adaptive Edge Detection.
- bool OverlayImage (HImage *image, HImage *overlayImage, int r, int g, int b)

Overlay detected edges image with specified RGB color triplet.

- bool MoveTowards (HImage *backgroundImage, HImage *foregroundImage, HImage *destImage)

  Move toward motion detection: perform an absolute subtraction in between the background and foreground image.
- bool ConvertToGray (HImage *image, HImage *grayImage)

  Convert an image to the Gray 8 format.
- bool MeanGrayDetect (HImage *image, double &value)

  Perform a Mean Gray Detection.
- bool AbsDiffDetect (HImage *image, HImage *background, double &value, double &mean)

  Perform an Absolute Difference detection.
- bool AbsMaxDiffDetect (HImage *image, HImage *background, double &max)

  Perform a Max Absolute Difference detection.
- bool EdgeDetect (HImage *image, double &value)

  Perform an Edge Detection.
- bool EdgeDetect (HImage *image, HImage *resultImage, double &value)

  Perform an Edge Detection and count the number of pixels in the motion detected area.
- bool AdaptiveEdgeDetect (HImage *image, HImage *background, double &value)

  Perform an Adaptive Edge Detection.
- bool AdaptiveEdgeDetect (HImage *image, HImage *background, HImage *resultImage, double &value)

  Perform an Adaptive Edge Detection.
- bool OverlayImage (HImage *image, HImage *overlayImage, int r, int g, int b)

  Overlay detected edges image with specified RGB color triplet.
- bool MoveTowards (HImage *backgroundImage, HImage *foregroundImage, HImage *destImage)

  Move toward motion detection: perform an absolute subtraction in between the background and foreground image.

3.72.1 Detailed Description

This is an utility class that allows to detect object motion in images.

If a color image is used as a source for being processed, it will first be converted to a monochrome image.

3.72.2 Member Function Documentation

3.72.2.1 void HMotionDetection::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.72.2.2 eHErrorCode HMotionDetection::GetLastError ( )

See also

HGrabModule::GetLastError

3.72.2.3 void HMotionDetection::SetMotionDetectionMode (eHMotionDetectionMode *mode*)

Set motion detection mode.

Parameters

in	*mode*	Motion detection mode.

3.72.2.4 eHMotionDetectionMode HMotionDetection::GetMotionDetectionMode ( )

Get Motion Detection Mode.

Returns

The motion detection mode selected.

3.72.2.5 void HMotionDetection::SetSobelDetectionMode (eHSobelMode *mode*)

Set Sobel operating detection mode.

Parameters

in	*mode*	Sobel detection mode.

3.72.2.6 eHSobelMode HMotionDetection::GetSobelDetectionMode ( )

Get Sobel detection mode.

Returns

Current Sobel operating mode.

3.72.2.7 void HMotionDetection::SetThresholdValue (double *value*)

Set detection threshold.

Parameters

in	*value*	Threshold value.

3.72.2.8 double HMotionDetection::GetThresholdValue ( )

Get detection threshold.

Returns

Return the current threshold value.

3.72.2.9 void HMotionDetection::SetAbsoluteThresholdValue (int *value*)

Set Absolute Max detection threshold.

Parameters

in	*value*	Threshold value.

3.72.2.10 int HMotionDetection::GetAbsoluteThresholdValue ( )

Get Absolute Max detection threshold.

Returns

Return the Asbsolute Threshold Value.

3.72.2.11 void HMotionDetection::SetRoi (int *offsetX,* int *offsetY,* int *sizeX,* int *sizeY*)

Set Detection ROI.

Parameters

in	*offsetX*	The horizontal offset of the region.
in	*offsetY*	The vertical offset of the region.
in	*sizeX*	The width of the region.
in	*sizeY*	The height of the region.

3.72.2.12 void HMotionDetection::GetRoi (int & *offsetX,* int & *offsetY,* int & *sizeX,* int & *sizeY*)

Get Detection ROI.

Parameters

out	*offsetX*	The horizontal offset of the region.
out	*offsetY*	The vertical offset of the region.
out	*sizeX*	The width of the region.
out	*sizeY*	The height of the region.

3.72.2.13 bool HMotionDetection::ConvertToGray (HImage * *image,* HImage * *grayImage*)

Convert an image to the Gray 8 format.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to process.
out	*grayImage*	Result image.

3.72.2.14 bool HMotionDetection::MeanGrayDetect (HImage * *image,* double & *value*)

Perform a Mean Gray Detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to process.
out	*value*	Average gray level value.

3.72.2.15 bool HMotionDetection::AbsDiffDetect (HImage * *image,* HImage * *background,* double & *value,* double & *mean*)

Perform an Absolute Difference detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to process.
in	*background*	Background image.
out	*value*	Number of pixels difference with background.
out	*mean*	Average gray level value.

3.72.2.16 bool HMotionDetection::AbsMaxDiffDetect (HImage * *image,* HImage * *background,* double & *max*)

Perform a Max Absolute Difference detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to process.
in	*background*	Background image.
out	*max*	The value of max gray level different with background.

3.72.2.17 bool HMotionDetection::EdgeDetect (HImage * *image,* double & *value*)

Perform an Edge Detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
out	*value*	The number pxiels of motion detected area.

3.72.2.18 bool HMotionDetection::EdgeDetect (HImage * *image,* HImage * *resultImage,* double & *value*)

Perform an Edge Detection and count the number of pixels in the motion detected area.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
out	*resultImage*	Result image.

out	*value*	The pixels count in the motion detected area.

3.72.2.19 bool HMotionDetection::AdaptiveEdgeDetect (HImage * *image,* HImage * *background,* double & *value*)

Perform an Adaptive Edge Detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
in	*background*	Background to subtract.
out	*value*	The number pixels detected in the motion area.

3.72.2.20 bool HMotionDetection::AdaptiveEdgeDetect (HImage * *image,* HImage * *background,* HImage * *resultImage,* double & *value*)

Perform an Adaptive Edge Detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
in	*background*	background to subtract.
out	*resultImage*	result image.
out	*value*	The number pxiels of motion detected area.

3.72.2.21 bool HMotionDetection::OverlayImage (HImage * *image,* HImage * *overlayImage,* int *r,* int *g,* int *b*)

Overlay detected edges image with specified RGB color triplet.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
out	*overlayImage*	Result overlayed image.

in	*r*	red component of overlay color.
in	*g*	green component of overlay color.
in	*b*	blue component of overlay color.

3.72.2.22 bool HMotionDetection::MoveTowards (HImage * *backgroundImage,* HImage * *foregroundImage,* HImage * *destImage*)

Move toward motion detection: perform an absolute subtraction in between the background and foreground image.

Destination image contains all the pixel changes in between foreground and background image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*backgroundImage*	back ground image.
in	*foregroundImage*	fore ground image.
out	*destImage*	result image.

3.72.2.23 void HMotionDetection::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.72.2.24 eHErrorCode HMotionDetection::GetLastError ( )

See also

HGrabModule::GetLastError

3.72.2.25 void HMotionDetection::SetMotionDetectionMode (eHMotionDetectionMode *mode*)

Set motion detection mode.

Parameters

in	*mode*	Motion detection mode.

3.72.2.26 eHMotionDetectionMode HMotionDetection::GetMotionDetectionMode ( )

Get Motion Detection Mode.

Returns

The motion detection mode selected.

3.72.2.27 void HMotionDetection::SetSobelDetectionMode (eHSobelMode *mode*)

Set Sobel operating detection mode.

Parameters

in	*mode*	Sobel detection mode.

3.72.2.28 eHSobelMode HMotionDetection::GetSobelDetectionMode ( )

Get Sobel detection mode.

Returns

Current Sobel operating mode.

3.72.2.29 void HMotionDetection::SetThresholdValue (double *value*)

Set detection threshold.

Parameters

in	*value*	Threshold value.

3.72.2.30 double HMotionDetection::GetThresholdValue ( )

Get detection threshold.

Returns

Return the current threshold value.

3.72.2.31 void HMotionDetection::SetAbsoluteThresholdValue (int *value*)

Set Absolute Max detection threshold.

Parameters

in	*value*	Threshold value.

3.72.2.32 int HMotionDetection::GetAbsoluteThresholdValue ( )

Get Absolute Max detection threshold.

Returns

Return the Asbsolute Threshold Value.

3.72.2.33 void HMotionDetection::SetRoi (int *offsetX,* int *offsetY,* int *sizeX,* int *sizeY*)

Set Detection ROI.

Parameters

in	*offsetX*	The horizontal offset of the region.
in	*offsetY*	The vertical offset of the region.
in	*sizeX*	The width of the region.
in	*sizeY*	The height of the region.

3.72.2.34 void HMotionDetection::GetRoi (int & *offsetX,* int & *offsetY,* int & *sizeX,* int & *sizeY*)

Get Detection ROI.

Parameters

out	*offsetX*	The horizontal offset of the region.
out	*offsetY*	The vertical offset of the region.
out	*sizeX*	The width of the region.
out	*sizeY*	The height of the region.

3.72.2.35 bool HMotionDetection::ConvertToGray (HImage * *image,* HImage * *grayImage*)

Convert an image to the Gray 8 format.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to process.
out	*grayImage*	Result image.

3.72.2.36 bool HMotionDetection::MeanGrayDetect (HImage * *image,* double & *value*)

Perform a Mean Gray Detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to process.
out	*value*	Average gray level value.

3.72.2.37 bool HMotionDetection::AbsDiffDetect (HImage * *image,* HImage * *background,* double & *value,* double & *mean*)

Perform an Absolute Difference detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to process.
in	*background*	Background image.
out	*value*	Number of pixels difference with background.
out	*mean*	Average gray level value.

3.72.2.38 bool HMotionDetection::AbsMaxDiffDetect (HImage * *image,* HImage * *background,* double & *max*)

Perform a Max Absolute Difference detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to process.
in	*background*	Background image.
out	*max*	The value of max gray level different with background.

3.72.2.39 bool HMotionDetection::EdgeDetect (HImage * *image,* double & *value*)

Perform an Edge Detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
out	*value*	The number pxiels of motion detected area.

3.72.2.40 bool HMotionDetection::EdgeDetect (HImage * *image,* HImage * *resultImage,* double & *value*)

Perform an Edge Detection and count the number of pixels in the motion detected area.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
out	*resultImage*	Result image.

out	*value*	The pixels count in the motion detected area.

3.72.2.41 bool HMotionDetection::AdaptiveEdgeDetect (HImage * *image,* HImage * *background,* double & *value*)

Perform an Adaptive Edge Detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
in	*background*	Background to subtract.
out	*value*	The number pixels detected in the motion area.

3.72.2.42 bool HMotionDetection::AdaptiveEdgeDetect (HImage * *image,* HImage * *background,* HImage * *resultImage,* double & *value*)

Perform an Adaptive Edge Detection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
in	*background*	background to subtract.
out	*resultImage*	result image.
out	*value*	The number pxiels of motion detected area.

3.72.2.43 bool HMotionDetection::OverlayImage (HImage * *image,* HImage * *overlayImage,* int *r,* int *g,* int *b*)

Overlay detected edges image with specified RGB color triplet.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*image*	Source image to be detect.
out	*overlayImage*	Result overlayed image.

in	*r*	red component of overlay color.
in	*g*	green component of overlay color.
in	*b*	blue component of overlay color.

3.72.2.44 bool HMotionDetection::MoveTowards (HImage * *backgroundImage,* HImage * *foregroundImage,* HImage * *destImage*)

Move toward motion detection: perform an absolute subtraction in between the background and foreground image.

Destination image contains all the pixel changes in between foreground and background image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Possible failure may be due to the fact that the source image format is not supported for that conversion.

Parameters

in	*backgroundImage*	back ground image.
in	*foregroundImage*	fore ground image.
out	*destImage*	result image.

3.73 HMP3File Class Reference

This class manages mp3 audio files.

```
#include <HMP3File.h>
```

Public Member Functions

- CHermesMP3 * **GetMP3File** ()

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

FILE CREATE/ADDDATA/CLOSE FUNCTIONS

- bool Create (LPCTSTR mp3Path)

  Creates an empty MP3 file on disk, getting reading for storing audio samples.
- bool SetAudioParam (int sampleRate, int bitRate, int mapLen=0, int *maps=NULL)

  Set audio stream parameters.
- bool AddData (HAudioData *data)

  Append the specified audio data into the current MP3 file.
- bool EndRecording ()

  Finish audio recording and switch to playback mode.
- bool Open (LPCTSTR mp3Path)

  open a exist MP3 file on disk, getting playback audio samples.
- bool GetAudioInfo (int &sampleRate, int &channels, int &bitDepth, int &bitrate, int &samplesPerBuffer)

  Get audio stream parameters.
- unsigned int GetTotalAudioDatas ()

  Retrieve the number of audio blocks stored in the current MP3.

- bool ReadAudio (HAudioData *audioData)

  Read an audio data from the current MP3.
- bool GetTimeStamp (unsigned __int64 index, long &timeStamp, unsigned short &timeStampMS, unsigned short &timeStampUS)

  Retrieve the timestamp of a given frame in the sequence.
- unsigned __int64 GetPositionByTimeStamp (long timeStamp, unsigned short timeStampMS, unsigned short timeStampUS)

  Search the audio file to find the last audio that had been captured at a given time.
- bool SetReadingPosition (unsigned __int64 position)

  Set the start position of the read.
- unsigned __int64 GetReadingPosition ()

  Retrieves the current read position, by returning sequence index of the current read position.
- bool Close ()

  Closes an open MP3 File.

3.73.1 Detailed Description

This class manages mp3 audio files.

The typical (simplified) call order is :

```
Saving captured audio to a mp3 file on disk
HAACFile mp3;
mp3.Create(_T("c:\\audio.mp3"));
mp3.SetAudioParam(48000,128000);
mp3.AddData(audioDataA);
mp3.AddData(audioDataB);
mp3.AddData(audioDataC);
mp3.AddData(audioDataD);
mp3.AddData(audioDataE);
//and so on...
mp3.Close();
```

3.73.2 Member Function Documentation

3.73.2.1 void HMP3File::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.73.2.2 eHErrorCode HMP3File::GetLastError ( )

See also

HGrabModule::GetLastError

3.73.2.3 bool HMP3File::Create (LPCTSTR *mp3Path*)

Creates an empty MP3 file on disk, getting reading for storing audio samples.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*mp3Path*	The full path and file name where the audio is to be saved (ex: c:/audio.mp3). Any mp3 file with the same name will be overwritten.

3.73.2.4 bool HMP3File::SetAudioParam (int *sampleRate*, int *bitRate*, int *mapLen* = 0, int * *maps* = NULL)

Set audio stream parameters.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*sampleRate*	Audio sample rate. Should match the audio capture device settings.
in	*bitRate*	MP3 compresion bit rate in bit per second.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to write to.

3.73.2.5 bool HMP3File::AddData (HAudioData * *data*)

Append the specified audio data into the current MP3 file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.73.2.6 bool HMP3File::EndRecording ( )

Finish audio recording and switch to playback mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.73.2.7 bool HMP3File::Open (LPCTSTR *mp3Path*)

open a exist MP3 file on disk, getting playback audio samples.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*mp3Path*	The full path and file name where the audio is to open (ex: c:/audio.mp3). Any mp3 file with the same name will be overwritten.

3.73.2.8 bool HMP3File::GetAudioInfo (int & *sampleRate,* int & *channels,* int & *bitDepth,* int & *bitrate,* int & *samplesPerBuffer*)

Get audio stream parameters.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*sampleRate*	audio sample rate. Should match audio device settings.
out	*channels*	number of audio channels.
out	*bitDepth*	audio bitDepth
out	*bitrate*	compression bit rate in bit per second.
out	*samplesPer↩ Buffer*	sample per each audio buffer

3.73.2.9 unsigned int HMP3File::GetTotalAudioDatas ( )

Retrieve the number of audio blocks stored in the current MP3.

Returns

The number of audio blocks in the MP3.

3.73.2.10 bool HMP3File::ReadAudio (HAudioData * *audioData*)

Read an audio data from the current MP3.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.73.2.11 bool HMP3File::GetTimeStamp (unsigned __int64 *index,* long & *timeStamp,* unsigned short & *timeStampMS,* unsigned short & *timeStampUS*)

Retrieve the timestamp of a given frame in the sequence.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The audio data index.
out	*timeStamp*	The absolute timestamp in seconds.
out	*timeStampMS*	The millisecond part of the timestamp.
out	*timeStampUS*	The microsecond part of the timestamp.

3.73.2.12 unsigned __int64 HMP3File::GetPositionByTimeStamp (long *timeStamp,* unsigned short *timeStampMS,* unsigned short *timeStampUS*)

Search the audio file to find the last audio that had been captured at a given time.

Returns

the audio data index.

Parameters

in	*timeStamp*	The absolute timestamp in seconds.
in	*timeStampMS*	The millisecond part of the timestamp.
in	*timeStampUS*	The microsecond part of the timestamp.

3.73.2.13 bool HMP3File::SetReadingPosition (unsigned __int64 *position*)

Set the start position of the read.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

position	index [in] The desired playback position.

3.73.2.14 unsigned __int64 HMP3File::GetReadingPosition ( )

Retrieves the current read position, by returning sequence index of the current read position.

Returns

The current read position.

3.73.2.15 bool HMP3File::Close ( )

Closes an open MP3 File.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.74 HMP4File Class Reference

This class manages MP4 files.

```
#include <HMP4File.h>
```

Public Member Functions

- CHermesMP4File * **GetMP4File** ()
- bool Create (LPCTSTR seqPath, bool audio, bool video)

  Creates an empty MP4 file on disk.
- bool SetVideoParam (int width, int height, int interval, int bitreate, double framerate, bool codec, eDeviceAcclType type=H_CPU_DEVICE_ONLY, int deviceIndex=-1)

  Set video stream parameters.
- bool SetAudioParam (int sampleRate, int bitRate, int mapLen=0, int *maps=NULL)

  Set audio stream parameters.
- bool PrepareWrite ()

  Initialize compressor for being ready to stream to file.
- bool IsSuggestPushVideo ()

  Check if possible to push video.
- bool AddFrame (HImage *image, double pts=-1)

  Append new frame.
- bool IsSuggestPushAudio ()

  Check if possible to push audio.
- bool AddAudio (HAudioData *data)

  Append new audio sample.
- bool FinalizeWrite ()

  Notify compressor that all data has beeen added, ready to close the file.
- bool EndRecording ()

  Finish audio recording and switch to playback mode.
- bool Close ()

  Closes an open MP4 File.
- bool SetCallbacks (HMovieFileCallbacks *userCallbacks, unsigned int userData=0, void *userPtr=NULL)

  Set the callback class used by the mp4 File.
- bool Open (LPCTSTR seqPath, bool audio)

  Opens an existing mp4 file on disk, automatically detecting the sequence format and opening it accordingly.
- bool GetVideoInfo (int &width, int &height, int &bitrate, double &framerate)

  Get video stream parameters.
- bool GetAudioInfo (int &sampleRate, int &channels, int &bitDepth, int &bitrate, int &samplesPerBuffer)

  Set audio stream parameters.
- bool Read (HImage *image)

  Read an image from the current MP4.
- void SetPlaybackPosition (unsigned int index)

  Set the start position of the playback.
- unsigned int GetPlaybackPosition ()

  Get the start position of the playback.
- unsigned int GetRecordingPosition ()

  Get the current position of the recording.
- bool GetIndexAt (long time, unsigned short timeMS, unsigned short timeUS, unsigned int &index)

  Retrieve the index at the specified time stamp.
- eHMovieMode GetActionMode ()

  Get the current MP4 mode (playback or recording).
- bool Read (HImage *image, unsigned int index)

  Read an image from the current MP4.
- unsigned int GetAllocatedFrames ()

  Retrieve the number of images stored in the current MP4.
- unsigned int GetTotalAudioDatas ()

Retrieve the number of audio blocks stored in the current MP4.

- double GetPlaybackFrameRate ()

  Retrieve the current MP4 playback frame rate.
- void SetPlaybackFrameRate (double rate)

  Set the MP4 playback frame rate.
- unsigned long GetKeyFrameByIndex (unsigned long index, bool forward)

  Get nearest key frame by index.
- unsigned int GetAudioPlaybackPosition ()

  Get the start audio position of the playback.
- bool Play ()

  Initiates playback of a MP4 from its current position.
- bool PlayFixedRate ()
- bool Stop ()

  Stop MP4 playback.
- bool IsPlaying ()

  Check if the AVI is currently playing.
- bool ReadAudio (HAudioData *audioData)

  Read an audio data from the current MP4.
- NORPIX_ONLY bool **DisableDecodeFrame** (bool enable)

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.74.1 Detailed Description

This class manages MP4 files.

```
Saving captured images to an video and audio .mp4 file on disk.
HAACFile mp4;
mp4.Create(_T("c:\\test.mp4"),true,true);
mp4.SetVideoParam(640,480,30,30,50000,true,H_264_ACCEL_QUICKSYNC);
mp4.SetAudioParam(48000,2,12800);
mp4.PrepareWrite();

if(mp4.IsSuggestPushVideo())AddFrame.AddData(ImageA);
if(mp4.IsSuggestPushAudio())AddFrame.AddAudio(audioDataA);

if(mp4.IsSuggestPushVideo())AddFrame.AddData(ImageB);
if(mp4.IsSuggestPushAudio())AddFrame.AddAudio(audioDataB);

if(mp4.IsSuggestPushVideo())AddFrame.AddData(ImageC);
if(mp4.IsSuggestPushAudio())AddFrame.AddAudio(audioDataC);
//and so on...
mp4.FinalizeWrite();
mp4.Close();
```

3.74.2 Member Function Documentation

3.74.2.1 void HMP4File::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.74.2.2 eHErrorCode HMP4File::GetLastError ( )

See also

HGrabModule::GetLastError

3.74.2.3 bool HMP4File::Create (LPCTSTR *seqPath,* bool *audio,* bool *video*)

Creates an empty MP4 file on disk.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*seqPath*	The full path and file name where the sequence is to be saved (ex: c:/stream.mp4). Any AAC file with the same name will be overwritten.
in	*audio*	audio stream needed.
in	*video*	video stream needed.

3.74.2.4 bool HMP4File::SetVideoParam (int *width,* int *height,* int *interval,* int *bitreate,* double *framerate,* bool *codec,* eDeviceAcclType *type* = H_CPU_DEVICE_ONLY, int *deviceIndex* = -1)

Set video stream parameters.

We only support two kinds of video codec: H.264 and MPEG4.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*width*	video size width.
in	*height*	video size height .
in	*interval*	video compression GOP size .
in	*bitreate*	video bit rate in bit per second.
in	*framerate*	video frame rate in image per second.
in	*codec*	video codec type. True: use H.264 codec; False: use MPEG4 codec.
in	*type*	H.264 acceleration type: software, quicksync or discreet GPU.
in	*deviceIndex*	device index for NVidia H.264 compression.

3.74.2.5 bool HMP4File::SetAudioParam (int *sampleRate,* int *bitRate,* int *mapLen* = 0, int * *maps* = NULL)

Set audio stream parameters.

We only support AAC audio compression.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*sampleRate*	audio sample rate. Should match audio device settings.
in	*bitRate*	compression bit rate in bit per second.

in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to write to.

3.74.2.6 bool HMP4File::PrepareWrite ( )

Initialize compressor for being ready to stream to file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.74.2.7 bool HMP4File::IsSuggestPushVideo ( )

Check if possible to push video.

This should be called before adding any new video frame the the file. If the function returns false, wait a few ms and try again.

3.74.2.8 bool HMP4File::AddFrame (HImage * *image*, double *pts* = -1)

Append new frame.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	video frame to be added to file.
in	*pts*	Presentation time, -1 mean use default presentation time based on frame time stamp.

3.74.2.9 bool HMP4File::IsSuggestPushAudio ( )

Check if possible to push audio.

This should be called before adding any new video frame the the file. If the function returns false, wait a few ms and try again.

3.74.2.10 bool HMP4File::AddAudio (HAudioData * *data*)

Append new audio sample.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*data*	audio data to be added.

3.74.2.11 bool HMP4File::FinalizeWrite ( )

Notify compressor that all data has beeen added, ready to close the file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.74.2.12 bool HMP4File::EndRecording ( )

Finish audio recording and switch to playback mode.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.74.2.13 bool HMP4File::Close ( )

Closes an open MP4 File.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.74.2.14 bool HMP4File::SetCallbacks (HMovieFileCallbacks * *userCallbacks*, unsigned int *userData* = 0, void * *userPtr* = NULL)

Set the callback class used by the mp4 File.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*userCallbacks*	The callbacks class to use. This will be a pointer to one the HMP4File.
in	*userData*	This parameter can be used to pass a value to be used in the user's callback functions.
in	*userPtr*	This parameter can be used to pass a pointer to be used the user's callback functions.

3.74.2.15 bool HMP4File::Open (LPCTSTR *seqPath*, bool *audio*)

Opens an existing mp4 file on disk, automatically detecting the sequence format and opening it accordingly.

Opening a sequence will reset the following settings to their default values : looping, playback speed, playback frame skip, playback range.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*seqPath*	The full path where the MP4 is located.
in	*audio*	Allow to open audio track.

3.74.2.16 bool HMP4File::GetVideoInfo (int & *width,* int & *height,* int & *bitrate,* double & *framerate*)

Get video stream parameters.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*width*	video size width.
out	*height*	video size height .
out	*bitrate*	video bit rate in bit per second.
out	*framerate*	video frame rate in image per second.

3.74.2.17 bool HMP4File::GetAudioInfo (int & *sampleRate,* int & *channels,* int & *bitDepth,* int & *bitrate,* int & *samplesPerBuffer*)

Set audio stream parameters.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*sampleRate*	audio sample rate. Should match audio device settings.
out	*channels*	number of audio channels.
out	*bitDepth*	audio bitDepth
out	*samplesPer↩ Buffer*	sample per each audio buffer

3.74.2.18 bool HMP4File::Read (HImage * *image*)

Read an image from the current MP4.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*image*	An HImage to hold the image data read from MP4.

3.74.2.19 void HMP4File::SetPlaybackPosition (unsigned int *index*)

Set the start position of the playback.

Parameters

index	index [in] The desired playback position.

3.74.2.20 bool HMP4File::GetIndexAt (long *time*, unsigned short *timeMS*, unsigned short *timeUS*, unsigned int & *index*)

Retrieve the index at the specified time stamp.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*time*	The time in time_t format.
in	*timeMS*	The milliseconds part.
in	*timeUS*	The microseconds part.
out	*index*	The frame index.

3.74.2.21 eHMovieMode HMP4File::GetActionMode ( )

Get the current MP4 mode (playback or recording).

This is useful as some functions can only be called when the MP4 file is in recording mode while some other can only be called while in playback mode.

Returns

The current mode.

3.74.2.22 bool HMP4File::Read (HImage * *image*, unsigned int *index*)

Read an image from the current MP4.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*image*	An HImage to hold the image data read from MP4

in	*index*	The requested image index. This ranges from 1 to the value retrieved with GetAllocatedFrames().

3.74.2.23 unsigned int HMP4File::GetAllocatedFrames ( )

Retrieve the number of images stored in the current MP4.

Returns

The number of images in the MP4.

3.74.2.24 unsigned int HMP4File::GetTotalAudioDatas ( )

Retrieve the number of audio blocks stored in the current MP4.

Returns

The number of audio blocks in the MP4.

3.74.2.25 void HMP4File::SetPlaybackFrameRate (double *rate*)

Set the MP4 playback frame rate.

Parameters

in	*rate*	The playback frame rate.

3.74.2.26 unsigned long HMP4File::GetKeyFrameByIndex (unsigned long *index*, bool *forward*)

Get nearest key frame by index.

Parameters

in	*index*	The frame index to search.
in	*forward*	search direction; true forward else backword

3.74.2.27 bool HMP4File::Play ( )

Initiates playback of a MP4 from its current position.

The current position can be retrieved and set with the GetPlaybackPosition and SetPlaybackPosition functions before calling Play. The playback is done according to the timestamp of each image. Images will be sent by the callbacks functions (see HSequencerCallbacks).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.74.2.28 bool HMP4File::PlayFixedRate ( )

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.74.2.29 bool HMP4File::Stop ( )

Stop MP4 playback.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.74.2.30 bool HMP4File::IsPlaying ( )

Check if the AVI is currently playing.

Only available while in playback mode.

Returns

True if the AVI is playing, false in all other cases (paused, stopped, etc)

3.74.2.31 bool HMP4File::ReadAudio (HAudioData * *audioData*)

Read an audio data from the current MP4.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.75 HMPEGTsFile Class Reference

This class manages Mpeg2 files Each frame is H.264 compressed Each audio sample is AAC encoded.

```
#include <HMPEGTsFile.h>
```

Public Member Functions

- bool Create (LPCTSTR seqPath, bool audio, bool video)

  Creates an empty MPEG2 .ts file on disk.
- bool SetVideoParam (int width, int height, int interval, int bitreate, double framerate, eDeviceAcclType type=H_CPU_DEVICE_ONLY, int deviceIndex=-1)

  Set video stream parameters.
- bool SetAudioParam (int sampleRate, int bitRate, int mapLen=0, int *maps=NULL)

  Set audio stream parameters.
- bool PrepareWrite ()

  Initialize compressor.
- bool IsSuggestPushVideo ()

  Check if compressor is ready for more video frame.
- bool AddFrame (HImage *image, double pts=-1)

  append a new frame to the file.
- bool IsSuggestPushAudio ()

  Check if compressor is ready for more audio sample data.
- bool AddAudio (HAudioData *data)

  Append new audio data.

- bool FinalizeWrite ()

  Notify compressor that all data has beeen pushed, ready to close file.
- bool EndRecording ()

  Finish audio recording.
- bool Close ()

  Closes an open MPEG2 .ts File.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.75.1 Detailed Description

This class manages Mpeg2 files Each frame is H.264 compressed Each audio sample is AAC encoded.

```
Saving captured images to a mpeg2 (.ts) file on disk
HAACFile ts;
ts.Create(_T("c:\\test.ts"),true,true);
ts.SetVideoParam(640,480,30,30,50000,true,H_264_ACCEL_QUICKSYNC);
ts.SetAudioParam(48000,2,12800);
ts.PrepareWrite();

if(ts.IsSuggestPushVideo())AddFrame.AddData(ImageA);
if(ts.IsSuggestPushAudio())AddFrame.AddAudio(audioDataA);

if(ts.IsSuggestPushVideo())AddFrame.AddData(ImageB);
if(ts.IsSuggestPushAudio())AddFrame.AddAudio(audioDataB);

if(ts.IsSuggestPushVideo())AddFrame.AddData(ImageC);
if(ts.IsSuggestPushAudio())AddFrame.AddAudio(audioDataC);
//and so on...
ts.FinalizeWrite();
ts.Close();
```

3.75.2 Member Function Documentation

3.75.2.1 void HMPEGTsFile::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.75.2.2 eHErrorCode HMPEGTsFile::GetLastError ( )

See also

HGrabModule::GetLastError

3.75.2.3 bool HMPEGTsFile::Create (LPCTSTR *seqPath,* bool *audio,* bool *video*)

Creates an empty MPEG2 .ts file on disk.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*seqPath*	The full path where the sequence is to be saved (ex: c:/stream.ts). Any .ts with the same name will be overwritten.
in	*audio*	audio stream needed.
in	*video*	video stream needed.

3.75.2.4 bool HMPEGTsFile::SetVideoParam (int *width*, int *height*, int *interval*, int *bitreate*, double *framerate*, eDeviceAcclType *type* = H_CPU_DEVICE_ONLY, int *deviceIndex* = -1)

Set video stream parameters.

We only support H.264.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*width*	video size width.
in	*height*	video size height.
in	*interval*	video compression GOP size.
in	*bitreate*	video bit rate in bits per second.
in	*framerate*	video frame rate.
in	*type*	H.264 compression type.
in	*deviceIndex*	device index for NVidia H.264 compression.

3.75.2.5 bool HMPEGTsFile::SetAudioParam (int *sampleRate*, int *bitRate*, int *mapLen* = 0, int * *maps* = NULL)

Set audio stream parameters.

We only support AAC audio compression.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*sampleRate*	audio sample rate from the audio input device in Hertz.
in	*bitRate*	AAC compression bit rate in bits per second.
in	*mapLen*	The number of items in "maps".
in	*maps*	An array of channels to write to.

3.75.2.6 bool HMPEGTsFile::PrepareWrite ( )

Initialize compressor.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.75.2.7 bool HMPEGTsFile::AddFrame (HImage * *image*, double *pts* = -1)

append a new frame to the file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	video frame to append to the file.
in	*pts*	Presentation time, -1 mean use default presentation time.

3.75.2.8 bool HMPEGTsFile::AddAudio (HAudioData * *data*)

Append new audio data.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*data*	audio data to append.

3.75.2.9 bool HMPEGTsFile::FinalizeWrite ( )

Notify compressor that all data has beeen pushed, ready to close file.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.75.2.10 bool HMPEGTsFile::EndRecording ( )

Finish audio recording.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.75.2.11 bool HMPEGTsFile::Close ( )

Closes an open MPEG2 .ts File.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.76 HRotation Class Reference

This is an utility class that can be used to handle image rotation.

```
#include <HRotation.h>
```

Public Member Functions

- void SetDestinationImageSize (int width, int height)

  Set the destination image width and heigtht size in pixels.
- void GetDestinationImageSize (int &width, int &height)

  Get the destination image size in pixels.
- void SetCenterCoordinate (int x, int y)

  Set rotation center coordinates.
- void GetCenterCoordinate (int &x, int &y)

  Return the coordinate for the center of rotation, in pixels.
- void SetRotateAngle (double angle)

  Set the angle of rotation.
- double GetRotateAngle ()

  Return the rotation angle in degrees.
- void SetInterpolateMode (eHRotationInterpolationMode mode)

  Set the pixel interpolation mode used during the calculation.
- eHRotationInterpolationMode GetInterpolateMode ()

  Get the current interpolation mode.
- void SetSmoothEdge (bool smooth)

  Enable or disable edge smoothing while performing the rotation.
- bool GetSmoothEdge ()

  Get the current edge smoothing mode.
- bool IsImageFormatSupported (HImage *image)

  Check if an image can be rotated.
- bool Rotate (HImage *srcImage, HImage *destImage)

  Perform the image rotation.
- void SetDestinationImageSize (int width, int height)

  Set the destination image width and heigtht size in pixels.
- void GetDestinationImageSize (int &width, int &height)

  Get the destination image size in pixels.
- void SetCenterCoordinate (int x, int y)

  Set rotation center coordinates.
- void GetCenterCoordinate (int &x, int &y)

  Return the coordinate for the center of rotation, in pixels.
- void SetRotateAngle (double angle)

  Set the angle of rotation.
- double GetRotateAngle ()

  Return the rotation angle in degrees.
- void SetInterpolateMode (eHRotationInterpolationMode mode)

  Set the pixel interpolation mode used during the calculation.
- eHRotationInterpolationMode GetInterpolateMode ()

  Get the current interpolation mode.
- void SetSmoothEdge (bool smooth)

  Enable or disable edge smoothing while performing the rotation.
- bool GetSmoothEdge ()

Get the current edge smoothing mode.

- bool IsImageFormatSupported (HImage *image)

  Check if an image can be rotated.

- bool Rotate (HImage *srcImage, HImage *destImage)

  Perform the image rotation.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()
- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.76.1 Detailed Description

This is an utility class that can be used to handle image rotation.

The rotation is performed via a multi-threaded algorithm, optimized for Intel-based architecture processors. Supported image formats are: Monochrome, raw Bayer, RGB, BGR and BRGX as well as YUV444, for all bit depths: 8, 10, 12, 14 and 16 bit. The destination image can be internally allocated if not provided. The destination image format will be maintained. Size in X and Y for the destination image must be specified at the calling time.

The calculation can be performed according to 3 different algorithms: Nearest pixel (fastest), Bilinear interpolation or Bicubic interpolation (slowest). It is also possible to enable an edge smoothing mode, where edges will be better maintained in the rotated image.

Special rotation case: When the center of rotation is set to [0,0] and the destination image X and Y size are swapped compared to the source image, a +/-90 degrees rotation is performed. The orientation of the rotation is defined by the angle parameter. The algorithm choice, as well as edge smoothing parameters, are ignored. Both, image size in X and Y must be multiple of 4. If this is not the case, the operation will be performed on the closest possible size, while remaining lines or pixels will be zeroed in the destination image.

Pixels in the destination image that do not have any corresponding pixels in the source image are zeroed. Rotated pixels that do not fit in the destination image are clipped.

3.76.2 Member Function Documentation

3.76.2.1 void HRotation::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.76.2.2 eHErrorCode HRotation::GetLastError ( )

See also

HGrabModule::GetLastError

3.76.2.3 void HRotation::SetDestinationImageSize (int *width,* int *height*)

Set the destination image width and heigtht size in pixels.

Parameters

in	*width*	The width of the destination image, in pixels.
in	*height*	The height of the destination image, in pixels.

3.76.2.4 void HRotation::GetDestinationImageSize (int & *width,* int & *height*)

Get the destination image size in pixels.

Returns

The destination image width and height in pixels.

Parameters

out	*width*	the destination image width in pixels.
out	*height*	the destination image height in pixels.

3.76.2.5 void HRotation::SetCenterCoordinate (int *x,* int *y*)

Set rotation center coordinates.

Make sure to set the center to 0,0 for 90 and -90 degrees rotation. Center position can be set outside of image boundaries.

Parameters

in	*x*	The X position of the center of rotation.
in	*y*	The Y position of the center of rotation.

3.76.2.6 void HRotation::GetCenterCoordinate (int & *x,* int & *y*)

Return the coordinate for the center of rotation, in pixels.

Parameters

out	*x*	position in X for the center of rotation, in pixels.
out	*y*	position in Y for the center of rotation, in pixels.

3.76.2.7 void HRotation::SetRotateAngle (double *angle*)

Set the angle of rotation.

The rotation angle must be specified in degrees, between 0.0 to +/-360.0.

Parameters

in	*angle*	The rotation angle in degrees.

3.76.2.8 double HRotation::GetRotateAngle ( )

Return the rotation angle in degrees.

Returns

Return the rotation angle in degrees.

3.76.2.9 void HRotation::SetInterpolateMode (eHRotationInterpolationMode *mode*)

Set the pixel interpolation mode used during the calculation.

Parameters

in	*mode*	The interpolation mode

3.76.2.10 eHRotationInterpolationMode HRotation::GetInterpolateMode ( )

Get the current interpolation mode.

Returns

The interpolation mode.

3.76.2.11 void HRotation::SetSmoothEdge (bool *smooth*)

Enable or disable edge smoothing while performing the rotation.

Edge Smoothing is more CPU demanding.

Parameters

in	*smooth*	True to enable, false to disable.

3.76.2.12 bool HRotation::GetSmoothEdge ( )

Get the current edge smoothing mode.

Returns

True is edge smoothing is enabled, false otherwise.

3.76.2.13 bool HRotation::IsImageFormatSupported (HImage * *image*)

Check if an image can be rotated.

Some image formats are not yet supported.

Returns

True if the image can be rotated, false otherwise.

Parameters

in	*image*	The image to be rotated.

3.76.2.14 bool HRotation::Rotate (HImage * *srcImage,* HImage * *destImage*)

Perform the image rotation.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*srcImage*	The source image to be rotated.
in,out	*destImage*	The resulting rotated image.

3.76.2.15 void HRotation::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.76.2.16 eHErrorCode HRotation::GetLastError ( )

See also

HGrabModule::GetLastError

3.76.2.17 void HRotation::SetDestinationImageSize (int *width,* int *height*)

Set the destination image width and heigtht size in pixels.

Parameters

in	*width*	The width of the destination image, in pixels.
in	*height*	The height of the destination image, in pixels.

3.76.2.18 void HRotation::GetDestinationImageSize (int & *width,* int & *height*)

Get the destination image size in pixels.

Returns

The destination image width and height in pixels.

Parameters

out	*width*	the destination image width in pixels.
out	*height*	the destination image height in pixels.

3.76.2.19 void HRotation::SetCenterCoordinate (int *x,* int *y*)

Set rotation center coordinates.

Make sure to set the center to 0,0 for 90 and -90 degrees rotation. Center position can be set outside of image boundaries.

Parameters

in	*x*	The X position of the center of rotation.
in	*y*	The Y position of the center of rotation.

3.76.2.20 void HRotation::GetCenterCoordinate (int & *x,* int & *y*)

Return the coordinate for the center of rotation, in pixels.

Parameters

out	*x*	position in X for the center of rotation, in pixels.
out	*y*	position in Y for the center of rotation, in pixels.

3.76.2.21 void HRotation::SetRotateAngle (double *angle*)

Set the angle of rotation.

The rotation angle must be specified in degrees, between 0.0 to +/-360.0.

Parameters

in	*angle*	The rotation angle in degrees.

3.76.2.22 double HRotation::GetRotateAngle ( )

Return the rotation angle in degrees.

Returns

Return the rotation angle in degrees.

3.76.2.23 void HRotation::SetInterpolateMode (eHRotationInterpolationMode *mode*)

Set the pixel interpolation mode used during the calculation.

Parameters

in	*mode*	The interpolation mode

3.76.2.24 eHRotationInterpolationMode HRotation::GetInterpolateMode ( )

Get the current interpolation mode.

Returns

The interpolation mode.

3.76.2.25 void HRotation::SetSmoothEdge (bool *smooth*)

Enable or disable edge smoothing while performing the rotation.

Edge Smoothing is more CPU demanding.

Parameters

in	*smooth*	True to enable, false to disable.

3.76.2.26 bool HRotation::GetSmoothEdge ( )

Get the current edge smoothing mode.

Returns

True is edge smoothing is enabled, false otherwise.

3.76.2.27 bool HRotation::IsImageFormatSupported (HImage * *image*)

Check if an image can be rotated.

Some image formats are not yet supported.

Returns

True if the image can be rotated, false otherwise.

Parameters

in	*image*	The image to be rotated.

3.76.2.28 bool HRotation::Rotate (HImage * *srcImage*, HImage * *destImage*)

Perform the image rotation.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*srcImage*	The source image to be rotated.
in,out	*destImage*	The resulting rotated image.

3.77 HSequence Class Reference

This class manages sequence files for recording and playback.

```
#include <HSequence.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

CALLBACKS RELATED FUNCTIONS

- bool SetCallbacks (HSequencerCallbacks *userCallbacks, unsigned int userData=0, void *userPtr=NULL)
  Set the callback class used by the sequencer module.

FILE CREATE/OPEN/CLOSE FUNCTIONS

- bool Create (LPCTSTR seqPath=NULL, eHCompression compression=H_COMPRESSION_NONE, unsigned long fixedSizeInByte=0, int quality=100, bool noBuffering=true)
  Creates an empty sequence file on disk or in RAM.
- bool Open (LPCTSTR seqPath)
  Opens an existing sequence file on disk, automatically detecting the sequence format and opening it accordingly.
- bool **OpenWithBuffering** (LPCTSTR seqPath)
- bool Close ()
  Closes an open sequence, writing the sequence header to the sequence file and closing it.

- bool IsSequenceOpen ()
  Check if there is a sequence open.
- bool IsRAMSequence ()
  Check if the current sequence is in RAM.

READ/WRITE FUNCTIONS

- bool Write (HImage *image)
  Write the specified image in the current sequence, which will be appended at the current recording position.
- void Flush ()
  Flush all pending frames to disk or RAM.
- bool Read (HImage *image, unsigned int index)
  Read an image from the current sequence (on disk or in RAM).
- void SetRecordingPosition (unsigned int index)
  Set where the next image written will be stored in the current sequence.
- unsigned int GetRecordingPosition ()
  Retrieve the position where the next image will be written in the current sequence.
- bool SupportRewrite ()
  Check if the current sequence supports rewrites.
- void DumpHeader ()
  Immediately write the file header to the file on disk.
- double GetFrameRate ()
  Retrieve the average frame rate for the current sequence, in FPS.
- void SetFrameRate (double framerate)
  Modifies the average frame rate of a sequence.
- unsigned int GetAllocatedFrames ()
  Retrieve the number of images stored in the current sequence.

IMAGE & TIMESTAMP SEARCH FUNCTIONS

- bool ReadTimestamp (unsigned int index, long &time, unsigned short &timeMS, unsigned short &timeUS)
  Retrieve the timestamp of a given frame in the sequence.
- bool GetIndexAt (long time, unsigned short timeMS, unsigned short timeUS, unsigned int &index)
  Search the sequence to find the last frame that had been captured at a given time.
- bool GetIndexNear (long time, unsigned short timeMS, unsigned short timeUS, unsigned int &index)
  Search the sequence to find the frame which is the nearest in time to the given time.

PRE-ALLOCATION FUNCTIONS

- bool PreAllocateImages (unsigned int imageSizeBytes, unsigned int count)
  Pre-allocate images for use in a RAM sequence.
- unsigned int GetPreAllocatedImages ()
  Retrieve the current number of pre-allocated images.

METADATA FUNCTIONS

- bool EnableMetadata (unsigned int metadataSize=0, unsigned int metadataInfoSize=0)
  Tell the sequence to any metadata bundled with the frames.

PLAYBACK FUNCTIONS

- void SetPlaybackPosition (unsigned int index)
  Set the start position of the playback.
- unsigned int GetPlaybackPosition ()
  Retrieves the current playback position, by returning sequence index of the current playback position.
- bool Play ()
  Initiates playback of a sequence from its current position.
- bool PlayFixedRate ()

Initiates playback of a sequence from its current position.

- bool IsPlayFixedRate ()

  Check if the playback is being done (or will be done) at a fixed rate.
- bool Stop ()

  Stop sequence playback.
- void SetPlaybackSpeed (double fps)

  Set the playback framerate to be used with PlayFixedRate.
- double GetPlaybackSpeed ()

  Retrieve the speed of the fixed rate playback.
- bool IsPlaybackLagging ()

  Check if a timestamped playback is lagging behind.
- void SetPlaybackFrameSkip (unsigned int frameSkip)

  Set the frame skip value used by PlayFixedRate.
- unsigned int GetPlaybackFrameSkip ()

  Retrieve the frame skip used with PlayFixedRate.
- void SetMaxPlaybackInterval (int ms)

  Set the maximum display interval between 2 frames while in playback.
- int GetMaxPlaybackInterval ()

  Retrieve the maximum playback interval used by the Play() function.
- bool IsPlaying ()

  Retrieve the playback state, i.e.
- void SetPlaybackRange (unsigned int start, unsigned int end)

  Restrict playback to a specific frame range.
- void GetPlaybackRange (unsigned int &start, unsigned int &end)

  Get the current range used by playback functions.

SYNCHRONISATION FUNCTIONS

- bool ResynchronizePlayback (long offset, unsigned short offsetMS)

  Put an offset on the start time reference of a timestamped sequence playback.
- bool ResynchronizePlayback (unsigned long index)

  Jump to the frame index provided and continue playback from there.
- bool GetNextPlaybackFrameTime (long &time, unsigned short &timeMS, unsigned short &timeUS)

  Get the time stamp of the next image that will be shown during a playback.

IMAGE INFO GETTERS

- unsigned int GetImageWidth ()

  Retrieve the width of the images stored in the sequence.
- unsigned int GetImageHeight ()

  Rretrieve the height of the images stored in the sequence.
- unsigned int GetImageBitDepth ()

  Retrieves the bit depth of the images stored in the current sequence.
- unsigned int GetImageBitDepthReal ()

  Retrieve the real bit depth of the images stored in the current sequence.
- unsigned int GetImageSizeBytes ()

  Retrieve the size in bytes of each image stored in the sequence.
- eHImageFormat GetImageFormat ()

  Retrieve the format of the images stored in the sequence.
- unsigned int GetTrueImageSize ()

  Retrieve the size, of each image in the sequence, in bytes.

COMPRESSION FUNCTIONS

- eHCompression GetSequenceCompression ()

  Retrieves the compression used by the sequence to store images.
- bool SetVideoSequenceParameter (int bitrate, int gop)

  Set Video Sequence bitrate and gop parameter Support H264,HEVC and MPEG4 stream only.
- bool SetVideoEncoderType (eDeviceAcclType type, int deviceIndex=-1)

Set Video Sequence Ecoder type, current support CPU,Intel and Nvidia or AMD.

- void SetCompressionBayerPatternType (eHBayerPatternOrigin pattern)

  Set the bayer pattern for the GPU lossy JPEG and Cayer encoders.
- eHBayerPatternOrigin GetCompressionBayerPattern ()

  Retrieves the compression bayer pattern.
- void SetCayerCompressionType (eHCayer_Compression compression)

  Set cayer encoder compression type.
- eHCayer_Compression GetCayerCompressionType ()

  Retrieves the cayer compression type.

COMPRESSED SEQUENCE INDEX FUNCTIONS

- void SetCompressSeqIndexType (bool bFile)

  Set compression index stream type True if seperate file or false if NTFS stream.
- bool GetCompressSeqIndexType ()

  Retrieves compression index stream type.
- void SetAutoRecoveryIndexStream (bool bAuto)

  Set enable or disable auto discover index stream when open compressed SEQ file True if enable or false if disable.
- bool GetAutoRecoveryIndexStream ()

  Get mode auto discover index stream mode.
- virtual bool RecoverIndexStream (LPCTSTR seqPath)

  Force to discover compression index stream.

DESCRIPTION HANDLING FUNCTIONS

- bool GetDescription (BYTE *description)

  Retrieve the 512 bytes of custom data included in the sequence header.
- bool SetDescription (BYTE *description)

  Set the sequence description.
- eHDescriptionFormat GetDescriptionFormat ()

  Retrieve the format of the description written in the sequence header.
- void SetDescriptionFormat (eHDescriptionFormat format)

  Set the format of the data in the sequence description.

BAYER PATTERN FUNCTIONS

- bool IsBayer ()

  Get sequence bayer format, if any.
- void SetBayerPattern (eHBayerPatternOrigin pattern)

  Saves the pattern origin, used for a Bayer conversion, to the sequence header.
- bool GetBayerPattern (eHBayerPatternOrigin &pattern)

  Retrieve the pattern origin, from the sequence header.

REFERENCE FRAME FUNCTIONS

- void SetReferenceFrame (unsigned int index)

  Save the index of the frame to use as the reference.
- unsigned int GetReferenceFrame ()

  Retrieve the index of the reference frame for the current sequence.

REFERENCE TIME FUNCTIONS

If there is no reference time set, the time stamp of the first image of the sequence will be used as the reference time.

- bool HasReferenceTime ()

  Check if a sequence has a reference time.
- void SetReferenceTime (long time, unsigned short timeMS, unsigned short timeUS)

  Set the sequence's reference time.

- bool GetReferenceTime (long &time, unsigned short &timeMS, unsigned short &timeUS)

  Get the sequence's reference time.
- void ResetReferenceTime ()

  Remove the sequence's reference time.

GPU JPEG FUNCTIONS

- void SetJpegEncoderType (eJpegEncodeAlgoType type)

  Set Jpeg encoder type encoder type can be IPP(CPU),and GPU.
- eJpegEncodeAlgoType GetJpegEncodeType ()

  Get Jpeg encoder type /.
- void SetJpegDecoderType (eJpegDecodeAlgoType type)

  Set Jpeg decoder type encoder type can be IPP(CPU),Quicksync and TurobJpeg.
- eJpegDecodeAlgoType GetJpegDecodeType ()

  Get Jpeg decoder type /.
- void EnableGPUJpegCompressionBayerInterp (bool enable)

  Set enable or disable Bayer interpolation for RAW bayer image CUDA Jpeg compresssion.
- void EnableGPUJpegCompressionColorprcessingLUT (bool enable)

  Set enable or disable color procession for CUDA Jpeg compresssion True if enable or false if disable.
- bool UpdateGPUJpegCompressionLUT (HImageColorProcessing *processor)

  Update color procession lookup table for CUDA Jpeg compresssion.

MISC FUNCTIONS

- void Sort ()

  Sort each frame of a sequence by timestamp and copy them to an empty sequence.
- void Truncate (unsigned int index)

  Remove all frames from "index", up to the last frame of the sequence.
- bool HasMarkers ()

  Check if the sequence file has an associated marker file (.xml).
- void **SetAudioFileFlag** (eHAudioFileFormat format)
- eHAudioFileFormat **GetAudioFileFlag** ()
- void **SetDAQFileFlag** (bool enable)
- bool **HasDAQFileFlag** ()
- bool HasLTC ()

  Check if the images of the sequence have an embedded LTC time code.
- void SetTimeOffset (long timeOffsetUS)

  Set the time offset.
- long GetTimeOffset ()

  Get the time offset.
- void SetOldestFrameIndex (unsigned int index)

  Save the index of the oldest frame to the sequence header.
- unsigned int GetOldestFrameIndex ()

  Get the oldest frame index.

3.77.1 Detailed Description

This class manages sequence files for recording and playback.

The typical (simplified) call order is :

```
//Saving captured images to a sequence file on disk
HSequence sequence;
sequence.Create(_T("c:\\seqfile.seq"));
sequence.Write(imageA);
sequence.Write(imageB);
sequence.Write(imageC);
sequence.Write(imageD);
sequence.Write(imageE);
//and so on...
sequence.Close();
```

3.77.2 Member Function Documentation

3.77.2.1 void HSequence::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.77.2.2 eHErrorCode HSequence::GetLastError ( )

See also

HGrabModule::GetLastError

3.77.2.3 bool HSequence::SetCallbacks (HSequencerCallbacks * *userCallbacks*, unsigned int *userData* = 0, void * *userPtr* = NULL)

Set the callback class used by the sequencer module.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*userCallbacks*	The callbacks class to use. This will be a pointer to one the user's H↩ SequencerCallbacks-derived class.
in	*userData*	This parameter can be used to pass a value to be used in the user's callback functions.
in	*userPtr*	This parameter can be used to pass a pointer to be used the user's callback functions.

3.77.2.4 bool HSequence::Create (LPCTSTR *seqPath* = NULL, eHCompression *compression* = H_COMPRESSION_NONE, unsigned long *fixedSizeInByte* = 0, int *quality* = 100, bool *noBuffering* = true)

Creates an empty sequence file on disk or in RAM.

It will also reset the following settings to their default values : looping, playback speed, playback frame skip, playback range.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*seqPath*	The full path where the sequence is to be saved (ex: c:/capture.seq). Any sequence with the same name will be overwritten. If seqPath is empty, the sequence is created in RAM.

in	*compression*	The compression used to store images in the sequence. Default is raw uncompressed images. See other formats supported in HEnums.h.
in	*fixedSizeInByte*	Is used to force a constant image size for compressed sequences. The default value, 0, means that no fixed size is imposed. Warning : Forcing a fixed size will truncate images which are larger than fixedSize.
in	*quality*	The format quality settings (used by the H_COMPRESSION_JPEG). An higher quality incurs less image degradation but less compression.
in	*noBuffering*	Write directly to the disk skipping the Windows buffering (applies only to uncompressed sequences)

3.77.2.5 bool HSequence::Open (LPCTSTR *seqPath*)

Opens an existing sequence file on disk, automatically detecting the sequence format and opening it accordingly.

Opening a sequence will reset the following settings to their default values : looping, playback speed, playback frame skip, playback range.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*seqPath*	The full path where the sequence is located. (ex : c:/capture.seq).

3.77.2.6 bool HSequence::Close ( )

Closes an open sequence, writing the sequence header to the sequence file and closing it.

RAM sequence are permanently lost upon closure.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.77.2.7 bool HSequence::IsSequenceOpen ( )

Check if there is a sequence open.

Returns

True if a sequence is currently loaded, false otherwise.

3.77.2.8 bool HSequence::IsRAMSequence ( )

Check if the current sequence is in RAM.

Returns

True if the sequence is stored in RAM, false otherwise.

3.77.2.9 bool HSequence::Write (HImage * *image*)

Write the specified image in the current sequence, which will be appended at the current recording position.

Using SetRecordingPosition will set a specific recording position, overwriting previously recorded images. Once the function completes, the recording position will be set to append the next image to the current one.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The image to be saved to the sequence.

3.77.2.10 bool HSequence::Read (HImage * *image*, unsigned int *index*)

Read an image from the current sequence (on disk or in RAM).

An Himage will need to be allocated to be read and freed once it is done with. Refer to the example below. Example :

```
HSequence sequence;
//Creates or opens a sequence file.  The sequence must not be empty.
...

//Allocate a HImage to hold the image that will be read.
HImage* image = HImage::CreateHImage();

//Read the first image of the sequence
sequence.Read(image, 1);

//Display the image or processes it
...

//Deallocate the HImage
HImage::DeleteHImage(image);
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*image*	An HImage to hold the image data read from the sequence.
in	*index*	The requested image index. This ranges from 1 to the value retrieved with GetAllocatedFrames().

3.77.2.11 void HSequence::SetRecordingPosition (unsigned int *index*)

Set where the next image written will be stored in the current sequence.

Some sequence formats, such as JPEG, will not allow the overwriting of previous images. SetRecordingPosition will do nothing if that is the case. Example :

```
//Recording in a 10 images loop

HSequence sequence;
sequence.Create("C:\\sequence.seq");

***

//Call SaveImage when an image is received
```

```
void SaveImage(HImage* image)
{
    //Write the image to the sequence
       sequence.Write(image);

    //Once 10 images have been captured, continue capture the beginning
    if(sequence.GetRecordingPosition() == 10)
           sequence.SetRecordingPosition(1);
}
```

Parameters

in	*index*	The requested recording position.

3.77.2.12 unsigned int HSequence::GetRecordingPosition ( )

Retrieve the position where the next image will be written in the current sequence.

Returns

The current recording position.

3.77.2.13 bool HSequence::SupportRewrite ( )

Check if the current sequence supports rewrites.

If not, SetRecordingPosition will not have any effect and recording will be done sequentially. RAM sequences always support rewrites. Some sequences using a compressed format might not support rewrites as the image size is variable, meaning that rewrite could overwrite part of the following frame(s) in the sequence.

Returns

True if the sequence supports rewrites.

3.77.2.14 void HSequence::DumpHeader ( )

Immediately write the file header to the file on disk.

Normally, the sequence header is only written to the sequence file when the sequence is closed. However, when capturing long or unsupervised sequences where power failures or system crashes could happen, DumpHeader can be periodically called as a safety net. DumpHeader writes the sequence header to the sequence file immediately, saving all frames captured until called and allowing later access to an interrupted recording session. This will not have any effect on a RAM sequence.

3.77.2.15 double HSequence::GetFrameRate ( )

Retrieve the average frame rate for the current sequence, in FPS.

This is the value written in the sequence file header.

Returns

The frame rate of the sequence.

3.77.2.16 void HSequence::SetFrameRate (double *framerate*)

Modifies the average frame rate of a sequence.

By default, the average frame rate of a sequence is set at 30 fps. However, depending on the capture rate of the application, this rate might need to be adjusted. This is the value written in the sequence file header but is not used anywhere else.

Parameters

framerate	frameRate [in] The recommended frame rate in frames per second.

3.77.2.17 unsigned int HSequence::GetAllocatedFrames ( )

Retrieve the number of images stored in the current sequence.

Returns

The number of images in the sequence.

3.77.2.18 bool HSequence::ReadTimestamp (unsigned int *index*, long & *time*, unsigned short & *timeMS*, unsigned short & *timeUS*)

Retrieve the timestamp of a given frame in the sequence.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*index*	The frame index. This value must be between 1 and the frame count.
out	*time*	The absolute timestamp in seconds.
out	*timeMS*	The millisecond part of the timestamp.
out	*timeUS*	The microsecond part of the timestamp.

3.77.2.19 bool HSequence::GetIndexAt (long *time*, unsigned short *timeMS*, unsigned short *timeUS*, unsigned int & *index*)

Search the sequence to find the last frame that had been captured at a given time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*time*	The absolute timestamp in seconds.
in	*timeMS*	The millisecond part of the timestamp.
in	*timeUS*	The microsecond part of the timestamp.
out	*index*	The frame index.

3.77.2.20 bool HSequence::GetIndexNear (long *time*, unsigned short *timeMS*, unsigned short *timeUS*, unsigned int & *index*)

Search the sequence to find the frame which is the nearest in time to the given time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*time*	The absolute timestamp in seconds.
in	*timeMS*	The millisecond part of the timestamp.
in	*timeUS*	The microsecond part of the timestamp.
out	*index*	The frame index.

3.77.2.21 bool HSequence::PreAllocateImages (unsigned int *imageSizeBytes*, unsigned int *count*)

Pre-allocate images for use in a RAM sequence.

This only works if the current sequence is an uncompressed RAM sequence. Try this is the normal capture to RAM is dropping frames.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*imageSizeBytes*	The size in bytes of the images that will be saved to the sequence. You can get this value from the HImage::GetImageSizeBytes().
in	*count*	The number of image to preallocate. Make sure you don't exceed your current amount of available RAM.

3.77.2.22 unsigned int HSequence::GetPreAllocatedImages ( )

Retrieve the current number of pre-allocated images.

As the pre-allocated images are used while recording, this value will goes down.

Returns

The number of pre-allocated images.

3.77.2.23 bool HSequence::EnableMetadata (unsigned int *metadataSize* = 0, unsigned int *metadataInfoSize* = 0)

Tell the sequence to any metadata bundled with the frames.

To enable metadata acquisition, call this function after creating the sequence (but BEFORE the first image is written), i.e. between calls to Create() and Write().

Returns

True if the function is successful or false if the function failed.

Parameters

in	*metadataSize*	The maximal size taken by the metadata of each image. If '0', the size will be allocated dynamicaly. However, dynamic storage prevent recording in loop.
in	*metadataInfo↩ Size*	The size of the metadata info section. Only used if you create your own metadata types.

3.77.2.24 void HSequence::SetPlaybackPosition (unsigned int *index*)

Set the start position of the playback.

See also

Play()

Parameters

index	index [in] The desired playback position.

3.77.2.25 unsigned int HSequence::GetPlaybackPosition ( )

Retrieves the current playback position, by returning sequence index of the current playback position.

Returns

The current playback position.

3.77.2.26 bool HSequence::Play ( )

Initiates playback of a sequence from its current position.

The current position can be retrieved and set with the GetPlaybackPosition and SetPlaybackPosition functions before calling Play. The playback is done according to the timestamp of each image. Images will be sent by the callbacks functions (see HSequencerCallbacks).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Example :

```
//Starting a playback from the beginning of the current sequence.
sequence.SetPlaybackPosition(1);
sequence.Play();
```

3.77.2.27 bool HSequence::PlayFixedRate ( )

Initiates playback of a sequence from its current position.

The current position can be retrieved and set with the GetPlaybackPosition and SetPlaybackPosition functions before calling Play. Here, the playback proceeds at a fixed speed, independent on the time stamps. The default playback speed is 30 fps, but can be changed to anything using SetPlaybackSpeed. Images will be sent by the callbacks functions (see HSequencerCallbacks).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code. Example :

```
//Starting a fixed rate playback from the beginning of the current sequence.
sequence.SetPlaybackPosition(1);
sequence.PlayFixedRate();
```

3.77.2.28 bool HSequence::IsPlayFixedRate ( )

Check if the playback is being done (or will be done) at a fixed rate.

If Play was called to start the playback, the function returns false. If PlayFixedRate was called to start the playback, the function returns true. If there currently is no playback running (IsPlaying returns false), the function will return the last known state.

Returns

True if the playback proceeds at the frame rate specified by the user. False if playback proceeds following the images timestamps.

3.77.2.29 bool HSequence::Stop ( )

Stop sequence playback.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.77.2.30 void HSequence::SetPlaybackSpeed (double *fps*)

Set the playback framerate to be used with PlayFixedRate.

The default value being 30 fps. This function can also be called at any time during a fixed rate playback to change speed on-the-fly. Specifying a negative value will play the sequence backward. This function has no impact on the Play function.

Parameters

in	*fps*	The playback speed, in frames/seconds. Positive for forwards playback, negative for backwards playback.

3.77.2.31 double HSequence::GetPlaybackSpeed ( )

Retrieve the speed of the fixed rate playback.

See also

SetPlaybackSpeed

Returns

The current playback speed in frames per second.

3.77.2.32 bool HSequence::IsPlaybackLagging ( )

Check if a timestamped playback is lagging behind.

Playback lag can occur if Hermes can't read frames fast enough from the disk. This only apply to timestamped playback because fixed rate playback will skip frames if it gets behind.

Returns

True if the playback is lagging, false otherwise.

3.77.2.33 void HSequence::SetPlaybackFrameSkip (unsigned int *frameSkip*)

Set the frame skip value used by PlayFixedRate.

This function can also be called at any time during a fixed rate playback to change the frame skip on-the-fly. For instance, if the starting playback position is 1 and the frameSkip is 1, you will get frames 1, 3, 5, 7, 9, etc. Default value is 0. This function has no impact on the Play function.

Parameters

in	*frameSkip*	The frame skip value.

3.77.2.34 unsigned int HSequence::GetPlaybackFrameSkip ( )

Retrieve the frame skip used with PlayFixedRate.

See also

SetPlaybackFrameSkip

Returns

The current frame skip value.

3.77.2.35 void HSequence::SetMaxPlaybackInterval (int *ms*)

Set the maximum display interval between 2 frames while in playback.

When a playback is done using the Play() function, the playback is done following the timestamps. So if two successive images are 10 seconds appart, Hermes will wait 10 seconds between reading each image. This can be a problem if the recording is not done continuously. For example, you start a recording, stop it, then restart it later. To work around this problem, Hermes uses a maximum playback interval value. For example, if you set this value to 1000 ms, the playback will proceed with, at most, a one second interval between each successive frames. Default value is 5000 ms.

Parameters

in	*ms*	The maximum playback interval value in milliseconds.

3.77.2.36 int HSequence::GetMaxPlaybackInterval ( )

Retrieve the maximum playback interval used by the Play() function.

Returns

The current maximum playback interval value in milliseconds.

3.77.2.37 bool HSequence::IsPlaying ( )

Retrieve the playback state, i.e.

playing or not. Example :

```
HSequence sequence;

void TogglePlayback()
{
   if(sequence.IsPlaying())
         sequence.Stop();
   else
         sequence.Play();
}
```

Returns

True if playback is running, false if playback stopped.

3.77.2.38 void HSequence::SetPlaybackRange (unsigned int *start*, unsigned int *end*)

Restrict playback to a specific frame range.

If playback is started outside of the start and end bounds, the current playback position will automatically snap to start. This only affects the Play and PlayFixedRate functions. When end is reached, if looping is enabled, playback will continue from start. Otherwise, playback will stop. To disable ranged playback, set both parameters to 0 (which are the default values).

Parameters

in	*start*	The first frame to be used in playback.
in	*end*	The last frame to be used in playback.

3.77.2.39 void HSequence::GetPlaybackRange (unsigned int & *start*, unsigned int & *end*)

Get the current range used by playback functions.

If start and end are 0, ranged playback is disabled.

Parameters

out	*start*	The first frame used in playback.
out	*end*	The last frame used in playback.

3.77.2.40 bool HSequence::ResynchronizePlayback (long *offset*, unsigned short *offsetMS*)

Put an offset on the start time reference of a timestamped sequence playback.

Sending a negative offset will make the playback accellerate. For example, a "-3 seconds" offset tells the playback that it is 3 second late and must do some catch-up. A positive offset will make the playback stalls for the specified amount of time. This function only works on timestamped playback (not on fixed rate playback).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*offset*	The offset to apply (in seconds).
in	*offsetMS*	The offset to apply (milliseconds part)

3.77.2.41 bool HSequence::ResynchronizePlayback (unsigned long *index*)

Jump to the frame index provided and continue playback from there.

This function only works on timestamped playback (not on manual playback).

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

index	The frame index to jump to

3.77.2.42 bool HSequence::GetNextPlaybackFrameTime (long & *time,* unsigned short & *timeMS,* unsigned short & *timeUS*)

Get the time stamp of the next image that will be shown during a playback.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

`out`	*time*	The time stamp of the next frame.
`out`	*timeMS*	The time stamp of the next frame (milliseconds).
`out`	*timeUS*	The time stamp of the next frame (microseconds).

3.77.2.43 unsigned int HSequence::GetImageWidth ( )

Retrieve the width of the images stored in the sequence.

Returns

The image width in pixels.

3.77.2.44 unsigned int HSequence::GetImageHeight ( )

Rretrieve the height of the images stored in the sequence.

Returns

The image height in pixel.

3.77.2.45 unsigned int HSequence::GetImageBitDepth ( )

Retrieves the bit depth of the images stored in the current sequence.

Returns

The image bit depth.

3.77.2.46 unsigned int HSequence::GetImageBitDepthReal ( )

Retrieve the real bit depth of the images stored in the current sequence.

Returns

The image valid bit depth.

3.77.2.47 unsigned int HSequence::GetImageSizeBytes ( )

Retrieve the size in bytes of each image stored in the sequence.

Returns

The image size in bytes.

3.77.2.48 eHImageFormat HSequence::GetImageFormat ( )

Retrieve the format of the images stored in the sequence.

Returns

The image format.

3.77.2.49 unsigned int HSequence::GetTrueImageSize ( )

Retrieve the size, of each image in the sequence, in bytes.

That size can be larger that the actual size of each image, due to alignment boundaries considerations. The space a sequence will take on disk can be estimated by multiplying this value by the number of images in the sequence.

Returns

The size of one image on disk.

3.77.2.50 eHCompression HSequence::GetSequenceCompression ( )

Retrieves the compression used by the sequence to store images.

Returns

The compression format.

3.77.2.51 bool HSequence::SetVideoSequenceParameter (int *bitrate,* int *gop*)

Set Video Sequence bitrate and gop parameter Support H264,HEVC and MPEG4 stream only.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*bitrate*	The bitrate for Video compression
in	*gop*	The gop for Video compression

3.77.2.52 bool HSequence::SetVideoEncoderType (eDeviceAcclType *type,* int *deviceIndex* = -1)

Set Video Sequence Ecoder type, current support CPU,Intel and Nvidia or AMD.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*type*	The H264 encoder type
in	*deviceIndex*	device index for Nvidia device

3.77.2.53 void HSequence::SetCompressionBayerPatternType (eHBayerPatternOrigin *pattern*)

Set the bayer pattern for the GPU lossy JPEG and Cayer encoders.

GPU lossy JPEG and Cayer encoder support Bayer interpolation for RAW bayer image.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*pattern*	bayer pattern

3.77.2.54 eHBayerPatternOrigin HSequence::GetCompressionBayerPattern ( )

Retrieves the compression bayer pattern.

Returns

bayer pattern.

3.77.2.55 void HSequence::SetCayerCompressionType (eHCayer_Compression *compression*)

Set cayer encoder compression type.

Parameters

in	*compression*	cayer compression type

3.77.2.56 eHCayer_Compression HSequence::GetCayerCompressionType ( )

Retrieves the cayer compression type.

Returns

cayer compresssion type.

3.77.2.57 void HSequence::SetCompressSeqIndexType (bool *bFile*)

Set compression index stream type True if seperate file or false if NTFS stream.

Parameters

in	*bFile*	index stream type

3.77.2.58 bool HSequence::GetCompressSeqIndexType ( )

Retrieves compression index stream type.

Returns

true if seperate file or false if NTFS stream

3.77.2.59 void HSequence::SetAutoRecoveryIndexStream (bool *bAuto*)

Set enable or disable auto discover index stream when open compressed SEQ file True if enable or false if disable.

Parameters

in	*bAuto*	auto dsicover mode

3.77.2.60 bool HSequence::GetAutoRecoveryIndexStream ( )

Get mode auto discover index stream mode.

Returns

True if enable auto discover mode or false if disable auto discover mode

3.77.2.61 virtual bool HSequence::RecoverIndexStream (LPCTSTR *seqPath*) [virtual]

Force to discover compression index stream.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*seqPath*	The SEQ file name to discover

3.77.2.62 bool HSequence::GetDescription (BYTE * *description*)

Retrieve the 512 bytes of custom data included in the sequence header.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*description*	The description from the sequence header will be written here. Parameter allocation is user selected and could be "BYTE description[512];".

3.77.2.63 bool HSequence::SetDescription (BYTE * *description*)

Set the sequence description.

The header of a sequence file comprises 512 bytes to be used at the user discretion. They can be used to store a description of the sequence, identifiers, related external data or anything else.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*description*	The description to be written to the sequence header. description must be allocated to hold at least 512 bytes. The 512 bytes will be copied as-is.

3.77.2.64 eHDescriptionFormat HSequence::GetDescriptionFormat ( )

Retrieve the format of the description written in the sequence header.

See also

SetDescriptionFormat

Returns

The format of the description.

3.77.2.65 void HSequence::SetDescriptionFormat (eHDescriptionFormat *format*)

Set the format of the data in the sequence description.

- H_UNICODE_FORMAT : The description is an UNICODE string that can be casted to a (wchar_t*)
- H_MBCS_FORMAT : The description is a MBCS string that can be casted to a (char*)
- H_BYTE_FORMAT : The description is binary data. Interpretation is left at user discretion.

Parameters

in	*format*	The format of the description.

3.77.2.66 bool HSequence::IsBayer ( )

Get sequence bayer format, if any.

Returns

True if a Bayer conversion can be applied on the current sequence, false otherwise.

3.77.2.67 void HSequence::SetBayerPattern (eHBayerPatternOrigin *pattern*)

Saves the pattern origin, used for a Bayer conversion, to the sequence header.

An application can use this value to store the right pattern origin, for later use.

Parameters

in	*pattern*	The pattern origin. See HEnums.h for all possible patterns.

3.77.2.68 bool HSequence::GetBayerPattern (eHBayerPatternOrigin & *pattern*)

Retrieve the pattern origin, from the sequence header.

If no pattern was set, the returned value is H_BAYER_PATTERNORIGIN_GB.

Returns

True if the sequence is Bayer, false otherwise. See IsBayer method for more details.

Parameters

out	*pattern*	The pattern origin.

3.77.2.69 void HSequence::SetReferenceFrame (unsigned int *index*)

Save the index of the frame to use as the reference.

An application can use this value to mark a specific frame in the sequence as the "event" of the sequence. For example, this value could be used to show time stamps relative to this frame. The reference frame is 0 by default (i.e. no reference frame)

Parameters

in	*index*	The index of the reference frame.

3.77.2.70 unsigned int HSequence::GetReferenceFrame ( )

Retrieve the index of the reference frame for the current sequence.

If no reference frame was defined, the returned value is 0.

Returns

The index of the reference frame.

3.77.2.71 bool HSequence::HasReferenceTime ( )

Check if a sequence has a reference time.

Returns

True if the sequence has a reference time.

3.77.2.72 void HSequence::SetReferenceTime (long *time*, unsigned short *timeMS*, unsigned short *timeUS*)

Set the sequence's reference time.

The reference time can a time value outside the time range of the sequence.

Parameters

in	*time*	The absolute reference time in seconds (32-bit time_t).
in	*timeMS*	The millisecond part of the reference time.
in	*timeUS*	The microsecond part of the reference time.

3.77.2.73 bool HSequence::GetReferenceTime (long & *time,* unsigned short & *timeMS,* unsigned short & *timeUS*)

Get the sequence's reference time.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*time*	The absolute reference time in seconds (32-bit time_t).
out	*timeMS*	The millisecond part of the reference time.
out	*timeUS*	The microsecond part of the reference time.

3.77.2.74 void HSequence::SetJpegEncoderType (eJpegEncodeAlgoType *type*)

Set Jpeg encoder type encoder type can be IPP(CPU),and GPU.

Parameters

in	*type*	Jpeg encoder type

3.77.2.75 eJpegEncodeAlgoType HSequence::GetJpegEncodeType ( )

Get Jpeg encoder type /.

Returns

Jpeg encoder type

3.77.2.76 void HSequence::SetJpegDecoderType (eJpegDecodeAlgoType *type*)

Set Jpeg decoder type encoder type can be IPP(CPU),Quicksync and TurobJpeg.

Parameters

in	*type*	Jpeg decoder type

3.77.2.77 eJpegDecodeAlgoType HSequence::GetJpegDecodeType ( )

Get Jpeg decoder type /.

Returns

Jpeg decoder type

3.77.2.78 void HSequence::EnableGPUJpegCompressionBayerInterp (bool *enable*)

Set enable or disable Bayer interpolation for RAW bayer image CUDA Jpeg compresssion.

True if enable or false if disable

Parameters

in	*enable*	mode for GPU bayer interpolation

3.77.2.79 void HSequence::EnableGPUJpegCompressionColorprcessingLUT (bool *enable*)

Set enable or disable color procession for CUDA Jpeg compresssion True if enable or false if disable.

Parameters

in	*enable*	mode for GPU color procession

3.77.2.80 bool HSequence::UpdateGPUJpegCompressionLUT (HImageColorProcessing * *processor*)

Update color procession lookup table for CUDA Jpeg compresssion.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.77.2.81 void HSequence::Sort ( )

Sort each frame of a sequence by timestamp and copy them to an empty sequence.

This is useful to save a sequence captured in a recording loop, where frame 1 is not necessarily the oldest frame. Once the copy is done, the current sequence will destroy itself, rename the created sequence file by its own name and automatically open it. Obviously, calling sort while in the middle of a recording or playback is not recommended. Sort can be a lengthy operation depending of the number of frames to be sorted and is not available for compressed sequence formats.

3.77.2.82 void HSequence::Truncate (unsigned int *index*)

Remove all frames from "index", up to the last frame of the sequence.

Frames located before "index" will not be deleted. If index is "1", the entire sequence will be cleared.

Parameters

in	*index*	The frame index to truncate at.

3.77.2.83 bool HSequence::HasMarkers ( )

Check if the sequence file has an associated marker file (.xml).

Returns

True if the sequence has a markers file.

3.77.2.84 bool HSequence::HasLTC ( )

Check if the images of the sequence have an embedded LTC time code.

Returns

True if an LTC time code is available.

3.77.2.85 void HSequence::SetTimeOffset (long *timeOffsetUS*)

Set the time offset.

The time offset is the time to add or remove to each frames time stamp. While this value isn't used internally, it can be used at the application level to apply a time offset to the timestamps read from a sequence. This is useful when the frames timestamps are off by a small/constant value.

Parameters

in	*timeOffsetUS*	The time offset (in microseconds).

3.77.2.86 void HSequence::SetOldestFrameIndex (unsigned int *index*)

Save the index of the oldest frame to the sequence header.

This function is provided for convenience only as this is done automatically every time an existing frame is overwritten.

Parameters

in	*index*	The frame index.

3.77.2.87 unsigned int HSequence::GetOldestFrameIndex ( )

Get the oldest frame index.

Useful for looped recording.

Returns

The frame index.

3.78 HSequencerCallbacks Class Reference

Allows the user to be notified when specific events occur in the HSequence.

```
#include <HSequencerCallbacks.h>
```

Public Member Functions

- virtual void OnImageWrite (unsigned int userData=0, void ∗userPtr=0)

  This function is called when an image was written to a sequence.
- virtual void OnImageRead (HImage ∗image, unsigned int userData=0, void ∗userPtr=0)

  This function is called when an image was just read from a sequence.
- virtual void OnQueryContinuePlayback (bool &continuePlayback, unsigned int userData=0, void ∗userPtr=0)

  This function is called after each image read during a playback to query if we want to interrupt the playback.

- virtual void OnEndOfSequence (bool &loop, unsigned int userData=0, void *userPtr=0)

  This function is called after the last image of the sequence is shown.
- virtual void OnEndOfPlaybackRange (bool &loop, unsigned int userData=0, void *userPtr=0)

  This function is called when the beginning/end of playback range is reached.
- virtual void OnPlaybackStopped (unsigned int userData=0, void *userPtr=0)

  This function is called when the sequence playback stops.

3.78.1 Detailed Description

Allows the user to be notified when specific events occur in the HSequence.

Make a class inheriting from HSequenceCallbacks and override the callback functions you want to act on.

3.78.2 Member Function Documentation

3.78.2.1 virtual void HSequencerCallbacks::OnImageWrite (unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called when an image was written to a sequence.

Parameters

out	*userData*	The value set in the SetCallback function.
out	*userPtr*	The value set in the SetCallback function.

3.78.2.2 virtual void HSequencerCallbacks::OnImageRead (HImage * *image*, unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called when an image was just read from a sequence.

Parameters

out	*image*	The image that was just read from the sequence file.
out	*userData*	The value set in the SetCallback function.
out	*userPtr*	The value set in the SetCallback function.

3.78.2.3 virtual void HSequencerCallbacks::OnQueryContinuePlayback (bool & *continuePlayback*, unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called after each image read during a playback to query if we want to interrupt the playback.

Parameters

in, out	*continue↩ Playback*	True to continue with the playback, false to stop it.
out	*userData*	The value set in the SetCallback function.
out	*userPtr*	The value set in the SetCallback function.

3.78.2.4 virtual void HSequencerCallbacks::OnEndOfSequence (bool & *loop*, unsigned int *userData* = 0, void * *userPtr* = 0) [inline], [virtual]

This function is called after the last image of the sequence is shown.

Parameters

in,out	*loop*	Set to true to loop to the begining of the sequence, false to stop playback.
out	*userData*	The value set in the SetCallback function.
out	*userPtr*	The value set in the SetCallback function.

3.78.2.5 virtual void HSequencerCallbacks::OnEndOfPlaybackRange (bool & *loop*, unsigned int *userData* = 0, void * *userPtr* = 0) [inline],[virtual]

This function is called when the beginning/end of playback range is reached.

Parameters

in,out	*loop*	Set to true to loop to the begining of the playback range, false to stop playback.
out	*userData*	The value set in the SetCallback function.
out	*userPtr*	The value set in the SetCallback function.

3.78.2.6 virtual void HSequencerCallbacks::OnPlaybackStopped (unsigned int *userData* = 0, void * *userPtr* = 0) [inline],[virtual]

This function is called when the sequence playback stops.

Parameters

out	*userData*	The value set in the SetCallback function.
out	*userPtr*	The value set in the SetCallback function.

3.79 HStreamDeliver Class Reference

This class is used to send StreamPix's captured audio or video to the streamer Do not inherited from this class, use HStreamServer::RegisterSourceUrl to get an instance pointer.

```
#include <HStreamEnums.h>
```

Public Member Functions

- virtual bool DeliverNewFrame (HImage *image, double timeOffset=0)=0

  This function is deliverring new video frame.
- virtual bool DeliverNewAudio (HAudioData *audio, double timeOffset=0)=0

  This function is deliverring new audio frame.
- virtual bool AddVirtualURL (char *url, char *username, char *password)=0

  This function is add virtual URL.
- virtual bool RemoveVirtualURL (char *url)=0

  This function is remove virtual URL.

3.79.1 Detailed Description

This class is used to send StreamPix's captured audio or video to the streamer Do not inherited from this class, use HStreamServer::RegisterSourceUrl to get an instance pointer.

3.79.2 Member Function Documentation

3.79.2.1 virtual bool HStreamDeliver::DeliverNewFrame (HImage * *image*, double *timeOffset* = 0) [pure virtual]

This function is deliverring new video frame.

Returns

true if successs

Parameters

in	*image*	source image.
in	*timeOffset*	time offset for video.

3.79.2.2 virtual bool HStreamDeliver::DeliverNewAudio (HAudioData * *audio*, double *timeOffset* = 0) [pure virtual]

This function is deliverring new audio frame.

Returns

true if successs

Parameters

in	*audio*	source audio data.
in	*timeOffset*	time offset for video.

3.79.2.3 virtual bool HStreamDeliver::AddVirtualURL (char * *url*, char * *username*, char * *password*) [pure virtual]

This function is add virtual URL.

Returns

true if successs

Parameters

in	*url*	virtual URL.
in	*username*	virtual user name.
in	*password*	virtual password.

3.79.2.4 virtual bool HStreamDeliver::RemoveVirtualURL (char * *url*) [pure virtual]

This function is remove virtual URL.

Returns

true if successs

Parameters

in	*url*	virtual URL.

3.80 HStreamServer Class Reference

Allows the user to create and manage web streaming using RTSP or HTTP live streaming protocol.

`#include <HStreamServer.h>`

Public Member Functions

CALLBACKS RELATED FUNCTIONS

Register a streaming URL.

Returns

HStreamDeliver instance pointer if success. Caller use this object to send video or audio data.

- HStreamDeliver * RegisterSourceUrl (char *url, HDeliverCallback *callback, int sampling, int gop=30, int bitrate=500000, eDeviceAcclType enoder=H_CPU_DEVICE_ONLY, int deviceIndex=0, eSTREAM_H264_METADATA medaData=H_STREAM_H264_METADATA_NONE, bool audio=false, int leftChannelOut=0, int rightChannelOut=1, int auidoBitrate=128000, bool resumeMode=false, bool alwaysEncoding=false)
- bool UnRegisterSourceUrl (char *url)

  Unregister a streaming URL.
- void SetHttpPort (int port)

  Set HTTP port.
- void SetRtspPort (int port)

  Set RTSP port.
- void SetRtspCredential (char *user, char *password)

  Set RTSP credential info.
- eSTREAM_ERROR StartServer (BYTE ip0, BYTE ip1, BYTE ip2, BYTE ip3, char *path, bool http Enable=false, bool hslEnable=false, bool rtspEnable=false, bool stateMonitoring=false)

  Start web streamer server. /.
- eSTREAM_ERROR **StopServer** ()

CALLBACKS RELATED FUNCTIONS.

- void SetCallback (HTTPRequest_Callback *callback, void *userData)

3.80.1 Detailed Description

Allows the user to create and manage web streaming using RTSP or HTTP live streaming protocol.

The video is compressed to H.264 and the audio is compressed to AAC. Current version supports audio streaming using RTSP protocol. No support for HTTP. The typical (simplified) call order is :

```
HStreamServer Server;                          //Instantiates a web streamer module.
Server.SetCallbacks(...);                      //Prepare the callbacks.
Server.SetRtspPort(...);                       //Set and reserve RTSP port.
HStreamDeliver* deliver;
 deliver = Server.RegisterSourceUrl(..., ...); //Register a streaming URL.
Server.StartServer();                          //Starts streamer server.
...                                            //Push video and or audio frames via
HStreamDeliver::DeliverNewFrame()
HStreamDeliver::DeliverNewAudio()
...
Server.StopServer();                           //Stop streamer server.
Server.UnRegisterSourceUrl();                  //Unregister the streaming driver.
```

3.80.2 Member Function Documentation

3.80.2.1 HStreamDeliver* HStreamServer::RegisterSourceUrl (char * *url*, HDeliverCallback * *callback*, int *sampling*, int *gop* = 30, int *bitrate* = 500000, eDeviceAcclType *enoder* = H_CPU_DEVICE_ONLY, int *deviceIndex* = 0, eSTREAM_H264_METADATA *medaData* = H_STREAM_H264_METADATA_NONE, bool *audio* = false, int *leftChannelOut* = 0, int *rightChannelOut* = 1, int *auidoBitrate* = 128000, bool *resumeMode* = false, bool *alwaysEncoding* = false)

Parameters

in	*url*	The URL to register.
in	*callback*	The callback for this channel in case some pre-processing is needed.
in	*sampling*	Video sampling interval.0,same as camera framerate.
in	*gop*	Video group size in number of frame
in	*bitrate*	Video H.264 bitrate in bit per second.
in	*enoder*	Video H.264 compression type. Software or use GPU acceleration.
in	*deviceIndex*	Nvidia device index for H.264 GPU compression.
in	*medaData*	Video H.264 metadata type (SEI).
in	*audio*	true if audio streaming is needed.
in	*leftChannelOut*	The left channel index for the audio.
in	*rightChannelOut*	The right channel index for the audio.
in	*auidoBitrate*	audio bitrate in bit per second.
in	*resumeMode*	Resume mode.If true reuse last video frame to stream
in	*alwaysEncoding*	True If always encode to H264 compression even input video contain H264 stream

3.80.2.2 bool HStreamServer::UnRegisterSourceUrl (char * *url*)

Unregister a streaming URL.

After the call this HStreamDeliver object will no longer be available.

Returns

true if success.

Parameters

in	*url*	The URL to unregister.

3.80.2.3 void HStreamServer::SetHttpPort (int *port*)

Set HTTP port.

Parameters

in	*port*	Port for HTTP.

3.80.2.4 void HStreamServer::SetRtspPort (int *port*)

Set RTSP port.

Parameters

in	*port*	Port for RTSP.

3.80.2.5 void HStreamServer::SetRtspCredential (char * *user*, char * *password*)

Set RTSP credential info.

Parameters

in	*user*	user name for RTSP credential.
in	*password*	Password for RTSP credential.

3.80.2.6 eSTREAM_ERROR HStreamServer::StartServer (BYTE *ip0*, BYTE *ip1*, BYTE *ip2*, BYTE *ip3*, char * *path*, bool *httpEnable* = `false`, bool *hslEnable* = `false`, bool *rtspEnable* = `false`, bool *stateMonitoring* = `false`)

Start web streamer server. /.

Returns

H_STREAM_NO_ERROR if success.

Parameters

in	*ip0*	The IPv4 address for first byte.
in	*ip1*	The IPv4 address for second byte.
in	*ip2*	The IPv4 address for third byte.
in	*ip3*	The IPv4 address for forth byte.
in	*path*	Home path for default HTTP.
in	*httpEnable*	True if enable http server.
in	*hslEnable*	True if enable http live streaming.
in	*rtspEnable*	True if enable rtsp.
in	*stateMonitoring*	True if monitor rtsp stream.

3.80.2.7 void HStreamServer::SetCallback (HTTPRequest_Callback * *callback*, void * *userData*)

Parameters

in	*callback*	The callback for HTTP request.
in	*userData*	Use this parameter to pass a pointer to be used in your callback functions.

3.81 HTcpCallbacks Class Reference

Manage TCP Callbacks.

`#include <HTcpCallbacks.h>`

Public Member Functions

- virtual void OnClientConnected (SOCKET socket, int error, void *userPtr)

  This callback function is called when a new connection is created.
- virtual void OnReceiveData (SOCKET socket, void *data, unsigned int dataSize, void *userPtr)

  This callback function is called when data is received from peer.

- virtual void OnClientDisconnected (SOCKET socket, int error, void *userPtr)

  This callback function is called when a connection is closed.

3.81.1 Detailed Description

Manage TCP Callbacks.

Make a class inheriting from HTcpCallbacks and override the callback functions you want to act on.

3.81.2 Member Function Documentation

3.81.2.1 virtual void HTcpCallbacks::OnClientConnected (SOCKET *socket*, int *error*, void * *userPtr*) [inline], [virtual]

This callback function is called when a new connection is created.

Parameters

out	*socket*	A descriptor identifying the connection socket.
out	*error*	The error code.
out	*userPtr*	The value set in the SetCallback function.

3.81.2.2 virtual void HTcpCallbacks::OnReceiveData (SOCKET *socket*, void * *data*, unsigned int *dataSize*, void * *userPtr*) [inline], [virtual]

This callback function is called when data is received from peer.

Parameters

out	*socket*	A descriptor identifying the connection socket.
out	*data*	A pointer to a buffer containing the received data.
out	*dataSize*	The length, in bytes, of the data in buffer pointed to by the data parameter.
out	*userPtr*	The value set in the SetCallback function.

3.81.2.3 virtual void HTcpCallbacks::OnClientDisconnected (SOCKET *socket*, int *error*, void * *userPtr*) [inline], [virtual]

This callback function is called when a connection is closed.

Parameters

out	*socket*	A descriptor identifying the closing socket.
out	*error*	The error code.
out	*userPtr*	The value set in the SetCallback function.

3.82 HTcpClient Class Reference

Manage a TCP client.

```
#include <HTcpClient.h>
```

Public Member Functions

- void SetCallback (HTcpCallbacks *callback, void *userPtr=NULL)

Set the callback class.

- bool ConnectTo (in_addr localAddress, in_addr remoteAddress, unsigned short port)

  Connect to a TCP server using the specified network interface identified by a local IP address.
- bool ConnectTo (BYTE localIp1, BYTE localIp2, BYTE localIp3, BYTE localIp4, BYTE remoteIp1, BYTE remoteIp2, BYTE remoteIp3, BYTE remoteIp4, unsigned short port)

  Connect to a TCP server using the specified network interface identified by a local IP address.
- bool IsConnected () const

  Check if the client is currently connected.
- bool Disconnect ()

  Close the current connection to the server.
- SOCKET GetSocket () const

  Retrieve the connection socket.
- bool Send (void *data, int dataSizeBytes)

  Send data over a connection.
- bool GetPeerAddress (in_addr &ipAddress, unsigned short &port)

  Retrieve the peer's IP address and communication port.
- bool GetPeerAddress (BYTE &ip1, BYTE &ip2, BYTE &ip3, BYTE &ip4, unsigned short &port)

  Retrieve the peer's IP address and communication port.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.82.1 Detailed Description

Manage a TCP client.

3.82.2 Member Function Documentation

3.82.2.1 void HTcpClient::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.82.2.2 eHErrorCode HTcpClient::GetLastError ( )

See also

HGrabModule::GetLastError

3.82.2.3 void HTcpClient::SetCallback (HTcpCallbacks * *callback*, void * *userPtr* = NULL)

Set the callback class.

See also

HTcpCallbacks.

Parameters

in	*callback*	A pointer to the user-defined callback class.
in	*userPtr*	A pointer to an user-defined buffer.

3.82.2.4 bool HTcpClient::ConnectTo (in_addr *localAddress,* in_addr *remoteAddress,* unsigned short *port*)

Connect to a TCP server using the specified network interface identified by a local IP address.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*localAddress*	The local IP address.
in	*remoteAddress*	The remote IP address to connect to.
in	*port*	The communication port to connect to.

3.82.2.5 bool HTcpClient::ConnectTo (BYTE *localIp1,* BYTE *localIp2,* BYTE *localIp3,* BYTE *localIp4,* BYTE *remoteIp1,* BYTE *remoteIp2,* BYTE *remoteIp3,* BYTE *remoteIp4,* unsigned short *port*)

Connect to a TCP server using the specified network interface identified by a local IP address.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*localIp1*	The first byte of the local IP address.
in	*localIp2*	The second byte of the local IP address.
in	*localIp3*	The third byte of the local IP address.
in	*localIp4*	The fourth byte of the local IP address.
in	*remoteIp1*	The first byte of the remote IP address.
in	*remoteIp2*	The second byte of the remote IP address.
in	*remoteIp3*	The third byte of the remote IP address.
in	*remoteIp4*	The fourth byte of the remote IP address.
in	*port*	The communication port.

3.82.2.6 bool HTcpClient::IsConnected ( ) const

Check if the client is currently connected.

Returns

True if the client is connected or false otherwise.

3.82.2.7 bool HTcpClient::Disconnect ( )

Close the current connection to the server.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.82.2.8 SOCKET HTcpClient::GetSocket ( ) const

Retrieve the connection socket.

Returns

If connected the function returns the existing connection socket or INVALID_SOCKET otherwise.

3.82.2.9 bool HTcpClient::Send (void * *data*, int *dataSizeBytes*)

Send data over a connection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*data*	A pointer to a buffer containing the data to be transmitted.
in	*dataSizeBytes*	The length, in bytes, of the data in buffer pointed to by the data parameter.

3.82.2.10 bool HTcpClient::GetPeerAddress (in_addr & *ipAddress*, unsigned short & *port*)

Retrieve the peer's IP address and communication port.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*ipAddress*	The peer's IP address.
out	*port*	The communication port.

3.82.2.11 bool HTcpClient::GetPeerAddress (BYTE & *ip1*, BYTE & *ip2*, BYTE & *ip3*, BYTE & *ip4*, unsigned short & *port*)

Retrieve the peer's IP address and communication port.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*ip1*	The first byte of the peer's IP address.
out	*ip2*	The second byte of the peer's IP address.
out	*ip3*	The third byte of the peer's IP address.
out	*ip4*	The fourth byte of the peer's IP address.
out	*port*	The communication port.

3.83 HTcpServer Class Reference

Manage a TCP server.

`#include <HTcpServer.h>`

Public Member Functions

- void SetCallback (HTcpCallbacks ∗callback, void ∗userPtr=NULL)

  Set the callback class (see HTcpCallbacks class).
- bool OpenServer (in_addr ipAddress, unsigned short port)

  Open a TCP server, listening for incoming connections.
- bool OpenServer (BYTE ip1, BYTE ip2, BYTE ip3, BYTE ip4, unsigned short port)

  Open a TCP server, listening for incoming connections.
- bool CloseServer ()

  Close the server, disconnecting all clients.
- bool IsServerOpen ()

  Check if the server is currently running.
- bool CloseConnection (SOCKET socket)

  Close an existing connection.
- void CloseAllConnections ()

  Close all connections.
- int GetConnectionCount ()

  Retrieve the total number of connections.
- bool GetAddress (in_addr &ipAddress, unsigned short &port)

  Retrieve server's IP address and port.
- bool GetAddress (BYTE &ip1, BYTE &ip2, BYTE &ip3, BYTE &ip4, unsigned short &port)

  Retrieve server's IP address and port.
- bool GetSocket (int connectionIndex, SOCKET &socket)

  Retrieve the socket of a connection identified by its index.
- SOCKET GetFirstSocket ()

  Retrieve the first socket in the connection list.
- SOCKET GetNextSocket ()

  Get the next socket in the connection list Retrieve the next socket in the connection list.
- bool Send (SOCKET socket, void ∗data, int dataSizeBytes)

  Send data to a connected socket.
- bool Broadcast (void ∗data, int dataSizeBytes)

  Send data to all connected sockets.
- bool GetPeerAddress (SOCKET socket, in_addr &ipAddress, unsigned short &port)

  Retrieve the peer's IP address and communication port.
- bool GetPeerAddress (SOCKET socket, BYTE &ip1, BYTE &ip2, BYTE &ip3, BYTE &ip4, unsigned short &port)

  Retrieve the peer's IP address and communication port.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.83.1 Detailed Description

Manage a TCP server.

3.83.2 Member Function Documentation

3.83.2.1 void HTcpServer::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.83.2.2 eHErrorCode HTcpServer::GetLastError ( )

See also

HGrabModule::GetLastError

3.83.2.3 void HTcpServer::SetCallback (HTcpCallbacks * *callback*, void * *userPtr* = NULL)

Set the callback class (see HTcpCallbacks class).

Parameters

in	*callback*	A pointer to the user-defined callback class.
in	*userPtr*	A pointer to an user-defined data.

3.83.2.4 bool HTcpServer::OpenServer (in_addr *ipAddress*, unsigned short *port*)

Open a TCP server, listening for incoming connections.

```
Example:
unsigned short port = 1234;
unsigned long ip = MAKEIPADDRESS(192, 168, 0, 1);
in_addr ipAddress; ipAddress.s_addr = htonl(ip);
TcpServer->OpenServer(ipAddress, port);
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*ipAddress*	The server's IP address.
in	*port*	The port the server uses to listen for new connections.

3.83.2.5 bool HTcpServer::OpenServer (BYTE *ip1*, BYTE *ip2*, BYTE *ip3*, BYTE *ip4*, unsigned short *port*)

Open a TCP server, listening for incoming connections.

```
Example:
unsigned short port = 1234;
unsigned long ip = MAKEIPADDRESS(192, 168, 0, 1);
in_addr ipAddress; ipAddress.s_addr = htonl(ip);
TcpServer->OpenServer(192, 168, 0, 1, port);
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*ip1*	The first byte of the server's IP address.
in	*ip2*	The second byte of the server's IP address.
in	*ip3*	The third byte of the server's IP address.
in	*ip4*	The fourth byte of the server's IP address.
in	*port*	The port the server uses to listen for new connections.

3.83.2.6 bool HTcpServer::CloseServer ( )

Close the server, disconnecting all clients.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.83.2.7 bool HTcpServer::IsServerOpen ( )

Check if the server is currently running.

Returns

True if the server is running or false otherwise.

3.83.2.8 bool HTcpServer::CloseConnection (SOCKET *socket*)

Close an existing connection.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*socket*	A descriptor identifying the socket to close.

3.83.2.9 int HTcpServer::GetConnectionCount ( )

Retrieve the total number of connections.

Returns

The total number of connections.

3.83.2.10 bool HTcpServer::GetAddress (in_addr & *ipAddress,* unsigned short & *port*)

Retrieve server's IP address and port.

Server must be open for this to work.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*ipAddress*	The server's IP address.
out	*port*	The communication port.

3.83.2.11 bool HTcpServer::GetAddress (BYTE & *ip1,* BYTE & *ip2,* BYTE & *ip3,* BYTE & *ip4,* unsigned short & *port*)

Retrieve server's IP address and port.

Server must be open for this to work.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*ip1*	The first byte of the server's IP address.
out	*ip2*	The second byte of the server's IP address.
out	*ip3*	The third byte of the server's IP address.
out	*ip4*	The fourth byte of the server's IP address.
out	*port*	The communication port.

3.83.2.12 bool HTcpServer::GetSocket (int *connectionIndex,* SOCKET & *socket*)

Retrieve the socket of a connection identified by its index.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*connectionIndex*	The zero-based connection index. Use GetConnectionCount to determine the total number of connections.
out	*socket*	The connection socket.

3.83.2.13 SOCKET HTcpServer::GetFirstSocket ( )

Retrieve the first socket in the connection list.

Returns

If successful, the function returns the first socket value in the connection list. In case of failure, the returned value is INVALID_SOCKET.

3.83.2.14 SOCKET HTcpServer::GetNextSocket ( )

Get the next socket in the connection list Retrieve the next socket in the connection list.

You must call GetFirstSocket before calling this function.

```
example :
SOCKET s = TcpServer->GetFirstSocket();
while(s != INVALID_SOCKET)
{
...
s = TcpServer->GetNextSocket();
}
```

Returns

If successful, the function returns the next socket value in the connection list. In case of failure or the end of the list is reached, the returned value is INVALID_SOCKET.

3.83.2.15 bool HTcpServer::Send (SOCKET *socket*, void * *data*, int *dataSizeBytes*)

Send data to a connected socket.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*socket*	A descriptor identifying a connected socket.
in	*data*	A pointer to a buffer containing the data to be transmitted.
in	*dataSizeBytes*	The length, in bytes, of the data in buffer pointed to by the data parameter.

3.83.2.16 bool HTcpServer::Broadcast (void * *data*, int *dataSizeBytes*)

Send data to all connected sockets.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*data*	A pointer to a buffer containing the data to be transmitted.
in	*dataSizeBytes*	The length, in bytes, of the data in buffer pointed to by the data parameter.

3.83.2.17 bool HTcpServer::GetPeerAddress (SOCKET *socket,* in_addr & *ipAddress,* unsigned short & *port*)

Retrieve the peer's IP address and communication port.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*socket*	A descriptor identifying a connected socket.
out	*ipAddress*	The peer's IP address.
out	*port*	The communication port.

3.83.2.18 bool HTcpServer::GetPeerAddress (SOCKET *socket,* BYTE & *ip1,* BYTE & *ip2,* BYTE & *ip3,* BYTE & *ip4,* unsigned short & *port*)

Retrieve the peer's IP address and communication port.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*socket*	A descriptor identifying a connected socket.
out	*ip1*	The first byte of the peer's IP address.
out	*ip2*	The second byte of the peer's IP address.
out	*ip3*	Third byte of the peer's IP address.
out	*ip4*	The fourth byte of the peer's IP address.
out	*port*	The communication port.

3.84 Http_Request Struct Reference

This structure contains information about the HTTP request.

```
#include <HStreamEnums.h>
```

Classes

- struct mg_header

Public Attributes

- char * **request_method**
- char * **uri**

- char * **http_version**
- char * **query_string**
- char * **remote_user**
- long **remote_ip**
- int **remote_port**
- int **status_code**
- int **is_ssl**
- int **num_headers**
- struct Http_Request::mg_header **http_headers** [64]

3.84.1 Detailed Description

This structure contains information about the HTTP request.

3.85 HTTPRequest_Callback Class Reference

Allows the user to be notified when an http request was made Make a class inheriting from HTTPRequest_Callback and override the callback functions you want to act on.

```
#include <HStreamCallback.h>
```

Public Member Functions

- virtual bool NewHttpRequest (const Http_Request *request, char **response, int *size, void *userData)=0

  This function is called upon a new http request.

3.85.1 Detailed Description

Allows the user to be notified when an http request was made Make a class inheriting from HTTPRequest_Callback and override the callback functions you want to act on.

3.85.2 Member Function Documentation

3.85.2.1 virtual bool HTTPRequest_Callback::NewHttpRequest (const Http_Request * *request*, char ** *response*, int * *size*, void * *userData*) [pure virtual]

This function is called upon a new http request.

Returns

true if the request has been processed

Parameters

in	*request*	Source image. For more details, view Http_Request in HStreamEnums.
out	*response*	The response buffer.
out	*size*	The response buffer size.
out	*userData*	The value set in the SetCallback function.

3.86 HUdpCallbacks Class Reference

Manage TCP Callbacks.

`#include <HUdpCallbacks.h>`

Public Member Functions

- virtual void OnReceiveData (SOCKET socket, void ∗data, unsigned int dataSize, const sockaddr_in ∗sender, void ∗userPtr)

  This callback function is called when data is received.

3.86.1 Detailed Description

Manage TCP Callbacks.

Make a class inheriting from HUdpCallbacks and override the callback functions you want to act on.

3.86.2 Member Function Documentation

3.86.2.1 virtual void HUdpCallbacks::OnReceiveData (SOCKET *socket*, void ∗ *data*, unsigned int *dataSize*, const sockaddr_in ∗ *sender*, void ∗ *userPtr*) [inline], [virtual]

This callback function is called when data is received.

Parameters

out	*socket*	A descriptor identifying the connection socket.
out	*data*	A pointer to a buffer containing the received data.
out	*dataSize*	The length, in bytes, of the data in buffer pointed to by the data parameter.
out	*sender*	A pointer to a buffer containing the address and port of the sender. May be NULL.
out	*userPtr*	The value set in the SetCallback function.

3.87 HUdpClient Class Reference

Manage a UDP client.

`#include <HUdpClient.h>`

Public Member Functions

- void SetCallback (HUdpCallbacks ∗callback, void ∗userPtr=NULL)

  Set the callback class.
- bool IsConnected () const

  Check if the UDP communication is currently active.
- bool Connect (bool useHeader, int options)

  Initiate an UDP communication.
- void Disconnect ()

  Close the communication.
- int SendTo (in_addr address, unsigned short port, const char ∗buffer, int bufferLen)

  Send data to a specific destination.
- int SendToEx (in_addr address, unsigned short port, const char ∗buffer, int bufferLen)

Send data to a specific destination.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.87.1 Detailed Description

Manage a UDP client.

3.87.2 Member Function Documentation

3.87.2.1 void HUdpClient::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.87.2.2 eHErrorCode HUdpClient::GetLastError ( )

See also

HGrabModule::GetLastError

3.87.2.3 void HUdpClient::SetCallback (HUdpCallbacks * *callback*, void * *userPtr* = NULL)

Set the callback class.

See also

HUdpCallbacks

Parameters

in	*callback*	A pointer to the user-defined callback class.
in	*userPtr*	A pointer to an user-defined data.

3.87.2.4 bool HUdpClient::IsConnected ( ) const

Check if the UDP communication is currently active.

Returns

True if the communication is active or false otherwise.

3.87.2.5 bool HUdpClient::Connect (bool *useHeader*, int *options*)

Initiate an UDP communication.

```
example :
UdpClient->Connect(true, SO_BROADCAST);//use header and enable transmission of broadcast messages.
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*useHeader*	If true, enables the datagram header transmission.
in	*options*	A parameter identifying the socket options. (e.g. SO_BROADCAST|SO_R↩ EUSEADDR)

3.87.2.6 int HUdpClient::SendTo (in_addr *address*, unsigned short *port*, const char ∗ *buffer*, int *bufferLen*)

Send data to a specific destination.

Returns

The total number of bytes sent.

Parameters

in	*address*	The destination IP address. For .NET version 0x7f000001 would be 127.0.0.1.
in	*port*	The destination port.
in	*buffer*	A pointer to a buffer containing the data to be transmitted.
in	*bufferLen*	The length, in bytes, of the data in buffer pointed to by the buffer parameter.

3.87.2.7 int HUdpClient::SendToEx (in_addr *address*, unsigned short *port*, const char ∗ *buffer*, int *bufferLen*)

Send data to a specific destination.

Each datagram has a header structure, described below:

```
struct CDatagramHeader
{
GUID guid;      //datagram ID
UINT dataSize;      //datagram size in bytes
UINT totalSize; //total number of bytes to be transmitted
UINT index;     //datagram index
UINT count;     //total number of datagrams to be transmitted
};
```

Returns

The total number of bytes sent.

Parameters

in	*address*	The destination IP address. For .NET version 0x7f000001 would be 127.0.0.1.
in	*port*	The destination port.
in	*buffer*	A pointer to a buffer containing the data to be transmitted.
in	*bufferLen*	The length, in bytes, of the data in buffer pointed to by the buffer parameter.

3.88 HUdpServer Class Reference

Manage a UDP server.

`#include <HUdpServer.h>`

Public Member Functions

- void SetCallback (HUdpCallbacks ∗callback, void ∗userPtr=NULL)

Set the callback class.

- bool IsOpened () const

  Check if the server is currently running.

- bool Open (in_addr ip, unsigned short port, bool useHeader, int options)

  Open a UDP server.

- void Close ()

  Close the server.

- bool GetIPAddress (in_addr *ip, unsigned short &port)

  Retrieve server's IP address and port.

- int SendTo (in_addr address, unsigned short port, const char *buffer, int bufferLen)

  Send data to a specific destination.

- int SendToEx (in_addr address, unsigned short port, const char *buffer, int bufferLen)

  Send data to a specific destination.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.88.1 Detailed Description

Manage a UDP server.

3.88.2 Member Function Documentation

3.88.2.1 void HUdpServer::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.88.2.2 eHErrorCode HUdpServer::GetLastError ( )

See also

HGrabModule::GetLastError

3.88.2.3 void HUdpServer::SetCallback (HUdpCallbacks * *callback*, void * *userPtr* = NULL)

Set the callback class.

See also

HUdpCallbacks

Parameters

in	*callback*	A pointer to the user-defined callback class.
in	*userPtr*	A pointer to an user-defined data.

3.88.2.4 bool HUdpServer::IsOpened ( ) const

Check if the server is currently running.

Returns

True if the server is running or false otherwise.

3.88.2.5 bool HUdpServer::Open (in_addr *ip*, unsigned short *port*, bool *useHeader*, int *options*)

Open a UDP server.

```
example :
unsigned short port = 1234;
unsigned long ip = MAKEIPADDRESS(192, 168, 0, 1);
in_addr ipAddr;
ipAddr.s_addr = htonl(ip);
UdpServer->Open(ipAddr, port, true, SO_BROADCAST|SO_REUSEADDR);
```

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*ip*	The server's IP address. For .NET version 0x7f000001 would be 127.0.0.1.
in	*port*	The port on which the server listens to.
in	*useHeader*	If true, enables the datagram header transmission.
in	*options*	The socket options (for example SO_BROADCAST).

3.88.2.6 bool HUdpServer::GetIPAddress (in_addr * *ip*, unsigned short & *port*)

Retrieve server's IP address and port.

Server must be open for this to work.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

out	*ip*	The server's IP address.
out	*port*	The communication port.

3.88.2.7 int HUdpServer::SendTo (in_addr *address*, unsigned short *port*, const char * *buffer*, int *bufferLen*)

Send data to a specific destination.

Returns

This function returns the total number of bytes sent.

Parameters

in	*address*	The destination IP address.
in	*port*	The destination port.
in	*buffer*	A pointer to a buffer containing the data to be transmitted.
in	*bufferLen*	The length, in bytes, of the data in buffer pointed to by the buffer parameter.

3.88.2.8 int HUdpServer::SendToEx (in_addr *address*, unsigned short *port*, const char * *buffer*, int *bufferLen*)

Send data to a specific destination.

Each datagram has a header structure, described below:

```
struct CDatagramHeader
{
   GUID guid;        //datagram ID
   UINT dataSize;    //datagram size in bytes
   UINT totalSize;   //total number of bytes to be transmitted
   UINT index;       //datagram index
   UINT count;       //total number of datagrams to be transmitted
};
```

Returns

This function returns the total number of bytes sent.

Parameters

in	*address*	The destination IP address.
in	*port*	The destination port.
in	*buffer*	A pointer to a buffer containing the data to be transmitted.
in	*bufferLen*	The length, in bytes, of the data in buffer pointed to by the buffer parameter.

3.89 HVideoOutputDriver Class Reference

Control a video output port.

`#include <HVideoOutputDriver.h>`

Public Member Functions

- bool IsVideoOutputAvailable ()

  Check if a specific video output port can be used.
- bool EnableVideoOutput ()

  Initialize a video output port.
- bool DisableVideoOutput ()

  Uninitialize a video output port.
- bool SendImageForVideoOutput (HImage *image)

  Send an image to an output port.

3.89.1 Detailed Description

Control a video output port.

To send frames to a supported external display.

3.89.2 Member Function Documentation

3.89.2.1 bool HVideoOutputDriver::IsVideoOutputAvailable ( )

Check if a specific video output port can be used.

Returns

True if the output port can be used.

3.89.2.2 bool HVideoOutputDriver::EnableVideoOutput ( )

Initialize a video output port.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.89.2.3 bool HVideoOutputDriver::DisableVideoOutput ( )

Uninitialize a video output port.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

3.89.2.4 bool HVideoOutputDriver::SendImageForVideoOutput (HImage * *image*)

Send an image to an output port.

The port must have been initialized using EnableVideoOutput().

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The image to send to the display port.

3.90 HViewer Class Reference

This class provides functions to draw HImages on-screen.

```
#include <HViewer.h>
```

Public Member Functions

- void SetZoom (int zoomX, int zoomY)

  Sets the horizontal and vertical scaling used when ShowImage is called to display an image.

- void GetZoom (int *zoomX, int *zoomY)

  Retrieves the horizontal and vertical scaling used when ShowImage is called to display an image.
- bool ShowImage (HDC deviceContext, HImage *image, int offsetX=0, int offsetY=0)

  Display an HImage on a user-created device context.
- bool ShowScaledImage (HDC deviceContext, HImage *image, int top, int left, unsigned int width, unsigned int height)

  Displays scaled HImage on an user-created device context.
- bool ShowImageOnDC (HDC deviceContext, HImage *image, RECT srcRect, RECT destRect, int zoomX=100, int zoomY=100, bool autoScaleUp=false, bool autoScaleDown=false, bool keepAspectRatio=true, RECT *resultRect=NULL)

  Display an HImage on a user-created device context, according to the parameters.

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

3.90.1 Detailed Description

This class provides functions to draw HImages on-screen.

Typical usage :

```
HViewer Viewer;

void CYourWindow::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    //LastImage must be a valid HImage
    Viewer.ShowImage(dc.m_hDC, LastImage);
}
```

3.90.2 Member Function Documentation

3.90.2.1 void HViewer::SetErrorOutput (bool *debugString,* bool *visual,* bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.90.2.2 eHErrorCode HViewer::GetLastError ( )

See also

HGrabModule::GetLastError

3.90.2.3 void HViewer::SetZoom (int *zoomX,* int *zoomY*)

Sets the horizontal and vertical scaling used when ShowImage is called to display an image.

This has no impact on ShowScaledImage or ShowImageOnDC. For example, if zoomX and zoomY are "200" (200%), the image will be shown two times bigger than its actual size. Showing a scaled image will require more CPU time. It is strongly recommended to leave the zooms at "100" (i.e. no scaling) when performance is required (when recording for example)

Parameters

in	*zoomX*	The horizontal scaling (%)
in	*zoomY*	The vertical scaling (%)

3.90.2.4 void HViewer::GetZoom (int * *zoomX*, int * *zoomY*)

Retrieves the horizontal and vertical scaling used when ShowImage is called to display an image.

Parameters

out	*zoomX*	The horizontal scaling (%)
out	*zoomY*	The vertical scaling (%)

3.90.2.5 bool HViewer::ShowImage (HDC *deviceContext*, HImage * *image*, int *offsetX* = 0, int *offsetY* = 0)

Display an HImage on a user-created device context.

Use the zoom functions to set the zoom level.

See also

SetZoom

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceContext*	Handle to the destination device context.
in	*image*	The image to be shown.
in	*offsetX*	The horizontal offset of the image on the canvas.
in	*offsetY*	The vertical offset of the image on the canvas.

3.90.2.6 bool HViewer::ShowScaledImage (HDC *deviceContext*, HImage * *image*, int *top*, int *left*, unsigned int *width*, unsigned int *height*)

Displays scaled HImage on an user-created device context.

The image is scaled to the specified size. This consumes more CPU time than ShowImage() due to the scaling required.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceContext*	Handle to the destination device context.
in	*image*	The image to be shown.

in	*top*	Specifies the y-coordinate of the upper-left corner of the destination rectangle.
in	*left*	Specifies the x-coordinate of the upper-left corner of the destination rectangle.
in	*width*	Specifies the width of the destination rectangle.
in	*height*	Specifies the height of the destination rectangle.

3.90.2.7 bool HViewer::ShowImageOnDC (HDC *deviceContext*, HImage * *image*, RECT *srcRect*, RECT *destRect*, int *zoomX* = 100, int *zoomY* = 100, bool *autoScaleUp* = false, bool *autoScaleDown* = false, bool *keepAspectRatio* = true, RECT * *resultRect* = NULL)

Display an HImage on a user-created device context, according to the parameters.

This consumes more CPU time if scaling is required.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*deviceContext*	Handle to the destination device context. (The DC to draw on)
in	*image*	The image to be shown.
in	*srcRect*	The part of the source image to display.
in	*destRect*	Where the drawing take place on the DC.
in	*zoomX*	Zoom percentage in X (ignored when autoScaleUp/autoScaleDown are enabled).
in	*zoomY*	Zoom percentage in Y (ignored when autoScaleUp/autoScaleDown are enabled).
in	*autoScaleUp*	True to scale image to fit destRect (if destRect > srcRect).
in	*autoScaleDown*	True to scale image to fit destRect (if destRect < srcRect).
in	*keepAspectRatio*	Keep image aspect ratio when scaling (ignore zoomY).
out	*resultRect*	Points to a user preallocated data that retrieves the result image rectangle. If NULL is passed, the parameter is ignored (The result image rectangle)

3.91 HWhiteBalance Class Reference

This class applies color balance for color image.

```
#include <HWhiteBalance.h>
```

Public Member Functions

ERROR MANAGEMENT FUNCTIONS

- void SetErrorOutput (bool debugString, bool visual, bool infoString)
- eHErrorCode GetLastError ()

//**CONFIGURATION FUNCTIONS**

- void GetColorBalance (int &red, int &green, int &blue)

  Get the color balance offset values applied in manual mode.
- void SetColorBalance (int red, int green, int blue)

  Set the color balance offset values applied when in manual mode.
- eHColorBalanceAlgorithm GetAlgorithm ()

  Get the automatic color balance (white balance) algorithm applied.

- void SetAlgorithm (eHColorBalanceAlgorithm value)
  Set the automatic color balance (white balance) algorithm.
- bool GetAutoColorBalanceMode ()
  Get the automatic color balance mode.
- void SetAutoColorBalanceMode (bool isAuto)
  Set the automatic color balance mode.

PROCESSING FUNCTIONS

- bool GetColorBalanceApplied ()
  Check if the color balance is applied on non-bayer images.
- void SetColorBalanceApplied (bool isColorBalanceApplied)
  Set color balance apply for non bayer image.
- bool ComputeColorBalance (HImage *image)
  Compute the automatic color balance.
- bool ComputeColorBalance (int red, int green, int blue)
  Compute the manual color balance.
- bool ApplyColorBalance (HImage *image)
  Apply the color balance in-place.
- bool ApplyColorBalance (HImage *inputImage, HImage *outputImage)
  Apply the color balance.
- bool ApplyColorBalance (HImage *inputImage, HImage *outputImage, int red, int green, int blue)
  Apply manual Color Balance.

3.91.1 Detailed Description

This class applies color balance for color image.

3.91.2 Member Function Documentation

3.91.2.1 void HWhiteBalance::SetErrorOutput (bool *debugString*, bool *visual*, bool *infoString*)

See also

HGrabModule::SetErrorOutput

3.91.2.2 eHErrorCode HWhiteBalance::GetLastError ( )

See also

HGrabModule::GetLastError

3.91.2.3 void HWhiteBalance::GetColorBalance (int & *red*, int & *green*, int & *blue*)

Get the color balance offset values applied in manual mode.

Parameters

out	*red*	Offset value applied to the red component.
out	*green*	Offset value applied to the green component.
out	*blue*	Offset value applied to the blue component.

3.91.2.4 void HWhiteBalance::SetColorBalance (int *red*, int *green*, int *blue*)

Set the color balance offset values applied when in manual mode.

Parameters

in	*red*	Offset value applied to the red component.
in	*green*	Offset value applied to the green component.
in	*blue*	Offset value applied to the blue component.

3.91.2.5 eHColorBalanceAlgorithm HWhiteBalance::GetAlgorithm ( )

Get the automatic color balance (white balance) algorithm applied.

Returns

The color balance algorithm.

3.91.2.6 void HWhiteBalance::SetAlgorithm (eHColorBalanceAlgorithm *value*)

Set the automatic color balance (white balance) algorithm.

Parameters

in	*value*	The algorithm to use.

3.91.2.7 bool HWhiteBalance::GetAutoColorBalanceMode ( )

Get the automatic color balance mode.

Returns

true auto balance mode; false manual color balance mode

3.91.2.8 void HWhiteBalance::SetAutoColorBalanceMode (bool *isAuto*)

Set the automatic color balance mode.

Parameters

in	*isAuto*	True for automatic color balance, false for manual color balance.

3.91.2.9 bool HWhiteBalance::GetColorBalanceApplied ( )

Check if the color balance is applied on non-bayer images.

Returns

true if the color balance is applied.

3.91.2.10 void HWhiteBalance::SetColorBalanceApplied (bool *isColorBalanceApplied*)

Set color balance apply for non bayer image.

Parameters

in	*isColor↩ BalanceApplied*	True to enable the color balancing, false to disable it.

3.91.2.11 bool HWhiteBalance::ComputeColorBalance (HImage * *image*)

Compute the automatic color balance.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*image*	The source image.

3.91.2.12 bool HWhiteBalance::ComputeColorBalance (int *red,* int *green,* int *blue*)

Compute the manual color balance.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*red*	Offset value applied to the red component.
in	*green*	Offset value applied to the green component.
in	*blue*	Offset value applied to the blue component.

3.91.2.13 bool HWhiteBalance::ApplyColorBalance (HImage * *image*)

Apply the color balance in-place.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in,out	*image*	The image to process.

3.91.2.14 bool HWhiteBalance::ApplyColorBalance (HImage * *inputImage,* HImage * *outputImage*)

Apply the color balance.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*inputImage*	The source image.
out	*outputImage*	The output image.

3.91.2.15 bool HWhiteBalance::ApplyColorBalance (HImage * *inputImage,* HImage * *outputImage,* int *red,* int *green,* int *blue*)

Apply manual Color Balance.

Returns

True if the function is successful or false if the function failed. In case of failure, immediately call GetLastError to retrieve the error code.

Parameters

in	*inputImage*	The source image.
out	*outputImage*	The output image.
in	*red*	Offset value applied to the red component.
in	*green*	Offset value applied to the green component.
in	*blue*	Offset value applied to the blue component.

3.92 Http_Request::mg_header Struct Reference

Public Attributes

- char * **name**
- char * **value**

Chapter 4

File Documentation

4.1 HEnums.h File Reference

Macros

- #define H_CHANNEL_LEFT 1

  Left channel.
- #define H_CHANNEL_RIGHT (1 << 1)

  Right channel.
- #define H_CHANNEL_STEREO (H_CHANNEL_LEFT | H_CHANNEL_RIGHT)

  Both channels.
- #define META_CODE(a, b, c, d) (((d) << 24)|((a) << 16) | (b<<8) | c)

  META_CODE.
- #define H_SEQBITFLAG_NULL 0x00

  No flags.
- #define H_SEQBITFLAG_HASMARKERS 0x01

  The sequence has an associated marker file.
- #define H_SEQBITFLAG_REFERENCE_TIME 0x02

  The sequence has a reference time.
- #define H_SEQBITFLAG_LTC 0x04

  The sequence has LTC code embedded in the images.
- #define H_SEQBITFLAG_H264_RGB 0x08

  ...
- #define H_SEQBITFLAG_METADATA 0x10

  The sequence has an associated metadata file.
- #define H_SEQBITFLAG_EAGLEEYE_MANUAL_TRIGGER 0x20

  ...
- #define H_SEQBITFLAG_HASAUDIO_FILE 0x40

  The sequence has an associated audio file.
- #define H_SEQBITFLAG_HASDAQ_FILE 0x80

  The sequence has an associated daq file.
- #define H_SEQBITFLAG_HASBWAVE_FILE 0x100

  The sequence has an associated audio file.
- #define H_SEQBITFLAG_HASMP3_FILE 0x200

  The sequence has an associated audio file.
- #define H_SEQBITFLAG_HASAAC_FILE 0x400

  The sequence has an associated audio file.

Enumerations

- enum eHSupportedModule {
  H_UNKNOWN = 0, H_ABSOFTWARE_ACTIVE_GIGE = 120, H_ABSOFTWARE_ACTIVE_USB = 159, H↩
  _ACTIVESILICON_FIREBIRD = 161,
  H_ACTIVESILICON_PHOENIX = 86, H_AJA_KONA = 168, H_ANDOR_LUCA = 92, H_ANDOR_NEO = 132,
  H_AVT_DCAM = 1, H_AVT_DCAM_UNI = 100, H_AVT_UNIVERSAL = 112, H_AXIS = 124,
  H_BASLER_IEEE1394 = 75, H_BASLER_GIGE = 77, H_BASLER_BCAM = 2, H_BASLER_PYLON = 150,
  H_BASLER_TOF = 173, H_BAUMER_FW = 90, H_BAUMER2 = 155, H_BITFLOW_CL = 3,
  H_BITFLOW_CXP = 164, H_BLACK_MAGIC = 134, H_CCAM = 167, H_CCDDIRECT_MICROPIX = 4,
  H_CMU_IEEE1394 = 5, H_COOKE_PCO = 6, H_COOKE_PIXELFLY = 7, H_COOKE_SENSICAM = 8,
  H_COOKE_SENSICAM_2 = 142, H_CORECO_BANDIT2CV = 9, H_CORECO_BANDIT2MV = 10, H_CO↩
  RECO_BANDIT2RGB = 11,
  H_CORECO_GIGE = 85, H_CORECO_VIPERDIG = 12, H_CORECO_PCCAMLINK = 13, H_CORECO_↩
  PC2CAMLINK = 14,
  H_CORECO_PC2COMP = 102, H_CORECO_PC2VISION = 15, H_CORECO_X64 = 16, H_DATAPATH_↩
  VISIONRGB = 113,
  H_EMERGENT = 145, H_EMS_PHYNXRGB = 17, H_EMS_XTREMERGB = 93, H_EMVACAM = 158,
  H_EPIPHAN = 133, H_EPIX_D2X = 18, H_EPIX_RAPTOR = 156, H_EPIX_SV5 = 19,
  H_EURESYS_GRABLINKVALUE = 22, H_EURESYS_IOTA = 23, H_EURESYS_PICOLO = 24, H_EURE↩
  SYS_PICOLO_HD = 154,
  H_PICOLO_H264_V16 = 114, H_PICOLO_H264_UX = 122, H_FCAV_ARINC818 = 97, H_FOCULUS_C↩
  AMERA = 25,
  H_FORESIGHTIMAGING_ACCUSTREAM = 87, H_FR_STREAM_USB = 162, H_GEVICAM = 99, H_GIG↩
  ALINX = 115,
  H_GIGALINX_SONY = 117, H_GRT_MATRIX = 103, H_GRT_MATRIX_HL2V = 125, H_HAMAMATSU_D↩
  CAM = 26,
  H_HASOTEC_FG30 = 27, H_HITACHI_DCAM = 28, H_HL2V = 98, H_IDS_UEYE = 83,
  H_IMAGINGSOURCE = 148, H_IMITECH_1394 = 89, H_IMITECH_GIGE = 116, H_IMITECH_NEPTUNE =
  143,
  H_IMPERX_BOBCAT_GIGE = 138, H_IMPERX_FRAMELINK = 29, H_IMPERX_FRAMELINK_EXPRESS =
  105, H_IMPERX_VCE_EXPRESS = 127,
  H_IMPERX_SDI_EXPRESS = 106, H_IMPERX_GIGE = 84, H_IMPERX_VCEPRO = 30, H_INTEGRAL_↩
  FLASHBUSDX = 31,
  H_INTEGRAL_FLASHBUSMV = 32, H_INTEGRAL_FLASHBUSMX = 33, H_INTEGRAL_SPECTRIM = 34,
  H_IP_CAMERA = 139,
  H_ISG_1394 = 21, H_JAIGENICAM_GIGE = 91, H_JAIGENICAM2 = 170, H_JAIPULNIX_GIGE = 74,
  H_JENOPTIK_DIJ = 169, H_JENOPTIK_PROGRESC10 = 35, H_JENOPTIK_PROGRESCXX = 36, H_JE↩
  NOPTIK_PROGRESC14 = 37,
  H_KAPPA_FW = 101, H_KAPPA_HIRES3XR = 153, H_KAYA = 176, H_KINECT = 141,
  H_LEUTRON_PICSIGHT_GIGE = 78, H_LEUTRON_SIMPLON_GIGE = 128, H_LUMENERA_GIGE = 121,
  H_LUMENERA_USB2 = 38,
  H_MATRIXVISION = 166, H_MATRIXVISION_BLUECOUGAR = 95, H_MATRIXVISION_BLUEFOX = 39,
  H_MATRIXVISION_HYPERION = 107,
  H_MATRIXVISION_SIGMA = 108, H_MATRIXVISION_TITAN = 109, H_MATROX_CXP = 151, H_MATR↩
  OX_CL = 152,
  H_MATROX_HELIOS = 40, H_MATROX_METEOR2_CL = 41, H_MATROX_METEOR2_DIG = 42, H_M↩
  ATROX_METEOR2_MC = 43,
  H_MATROX_METEOR2_STD = 44, H_MATROX_MORPHIS = 20, H_MATROX_RADIENT = 130, H_MA↩
  TROX_SOLIOS_XCL = 45,
  H_MATROX_ODYSSEY = 46, H_MATROX_ORION = 135, H_MATROX_VIO_XCL = 94, H_MIGHTEX =
  110,
  H_MIGHTEX_CCD = 126, H_NET_GIGE = 160, H_NET_ICUBE = 129, H_NI_GRABBERS_0 = 47,
  H_NIKON_D7000 = 140, H_NORPIX_VIRTUAL = 49, H_OLYMPUS_DP70 = 50, H_ONVIF = 131,
  H_OPTRONICS_MACROFIRE = 51, H_OPTRONICS_MAGNAFIRE = 52, H_OPTRONICS_MICROFIRE =
  53, H_PIXELINK_PLA500 = 54,
  H_PIXELINK_PLA600 = 55, H_PIXELINK_PLA700 = 56, H_PIXELINK_TM1020 = 57, H_PIXELINK_VC↩

A1281 = 58,
H_PIXERA_PENGUIN = 59, H_PLEORA_EBUS = 163, H_PLEORA_PT1000CL = 60, **H_PLEORA_MASTER** = 61,
H_PLEORA_SLAVE = 62, H_PRINCETON = 111, H_PROSILICA = 63, H_PROSILICA_GIGE = 64,
H_PTGREY = 119, H_PTGREY_LADYBUG = 149, H_QIMAGING_MP = 66, H_QIMAGING_RETIGA = 67,
H_REDLAKE_MEGAPLUSII = 68, H_ROPER_CASCADE = 69, H_ROPER_COOLSNAPCF = 70, H_SBF_FIREWIRE = 71,
H_SCION_CFW = 72, H_SILICONSOFTWARE_MICROENABLE = 123, H_SILICONSOFTWARE_MICROENABLE5 = 157, H_SILLICONIMAGING_GIGE = 73,
H_SPINNAKER = 171, H_SPOT = 76, H_SUMIX_12A2 = 96, H_SUMIX_150 = 79,
H_SUMIX_15M5X = 165, H_SUMIX_16EXX = 172, H_SUMIX_M7X = 80, H_SUMIX_M8X = 88,
H_SVS_VISTEK = 174, H_SVS_VISTEK_GIGE = 104, H_TATTILE_DUPLEX = 118, H_VIEWORKS = 144,
H_VIMBA = 147, H_VRMAGIC_RIC10 = 175, H_WDM_CAMERA = 82, H_XCELERA = 136,
H_XIMEA = 137, H_XENICS = 146 }

The supported cameras & frame grabbers.

- enum eHGrabberInfoType { H_GRABBER_INFO_TYPE_LONG = 0, H_GRABBER_INFO_TYPE_DOUBLE, H_GRABBER_INFO_TYPE_STRING }

  The type of grabber information.

- enum eHJpegQuality {
  H_JPEG_DEFAULT = 0, H_JPEG_FAST, H_JPEG_ACCURATE, H_JPEG_QUALITYSUPERB,
  H_JPEG_QUALITYGOOD, H_JPEG_QUALITYNORMAL, H_JPEG_QUALITYAVERAGE, H_JPEG_QUALITYBAD }

  The JPEG quality. Higher quality means slower encoding and bigger files.

- enum eHImageFormat {
  H_IMAGE_UNKNOWN = 0, H_IMAGE_MONO = 100, H_IMAGE_MONO_BAYER = 101, H_IMAGE_BGR = 200,
  H_IMAGE_PLANAR = 300, H_IMAGE_RGB = 400, H_IMAGE_BGRx = 500, H_IMAGE_YUV422 = 600,
  H_IMAGE_YUV422_20 = 610, **H_IMAGE_YUV422_PPACKED_10** = 620, H_IMAGE_YUV420_PLANAR = 630, H_IMAGE_UVY422 = 700,
  H_IMAGE_UVY411 = 800, H_IMAGE_UVY444 = 900, H_IMAGE_BGR555_PACKED = 905, H_IMAGE_BGR565_PACKED = 906,
  H_IMAGE_MONO_MSB = 112, H_IMAGE_MONO_BAYER_MSB = 113, H_IMAGE_MONO_MSB_SWAP = 114, H_IMAGE_MONO_BAYER_MSB_SWAP = 115,
  H_IMAGE_BGR10_PPACKED = 123, H_IMAGE_BGR10_PPACKED_PHOENIX = 124, H_IMAGE_RGB10_PPACKED_PHOENIX = 125, H_IMAGE_MONO_PPACKED = 131,
  H_IMAGE_MONO_BAYER_PPACKED = 132, H_IMAGE_MONO_PPACKED_8448 = 133, H_IMAGE_MONO_BAYER_PPACKED_8448 = 134, H_IMAGE_GVSP_BGR10V1_PACKED = 135,
  H_IMAGE_GVSP_BGR10V2_PACKED = 136, H_IMAGE_MONO_PPACKED_8844 = 137, H_IMAGE_MONO_BAYER_PPACKED_8844 = 138, **H_IMAGE_MONO_PPACKED_PCO** = 910,
  H_IMAGE_MONO_PPACKED_PCO_BAYER = 920, **H_IMAGE_BASLER_VENDOR_SPECIFIC** = 1000,
  H_IMAGE_EURESYS_JPEG = 1001, **H_IMAGE_ISG_JPEG** = 1002,
  H_IMAGE_PHOTON_FOCUS_DR_MONO = 1003, H_IMAGE_PHOTON_FOCUS_DR_BAYER = 1007, H_IMAGE_MONO80_64LSB_16MSB = 1005, H_IMAGE_GVSP_PIX_MONO10_PACKED = 1006,
  H_IMAGE_GVSP_PIX_MONO10_BAYER_PACKED = 1009, H_IMAGE_GRT_DUAL_IMAGE_RGB = 1008,
  H_IMAGE_PHOTON_FOCUS_QR_MONO = 1010, H_IMAGE_PHOTON_FOCUS_QR_BAYER = 1011,
  H_IMAGE_TOF_I_N_N = 4000, **H_IMAGE_TOF_I_RI16_N** = 4001, **H_IMAGE_TOF_I_RF32_N** = 4002, **H_IMAGE_TOF_I_N_II16** = 4003,
  H_IMAGE_TOF_I_RI16_II16 = 4004, **H_IMAGE_TOF_I_RF32_II16** = 4005 }

  The image format indicate how the pixel data is stored.

- enum eHCompression {
  H_COMPRESSION_NONE = 0, H_COMPRESSION_JPEG = 1, H_COMPRESSION_RLE = 2, H_COMPRESSION_HUFFMAN = 3,
  H_COMPRESSION_LZ = 4, H_COMPRESSION_RLE_FAST = 5, H_COMPRESSION_HUFFMAN_FAST = 6, H_COMPRESSION_LZ_FAST = 7,
  H_COMPRESSION_H264 = 8, H_COMPRESSION_WAVELET = 9, H_COMPRESSION_H264_RGB = 16,
  H_COMPRESSION_JPEG_LOSSLESS = 17,
  H_COMPRESSION_CAYER = 18, H_COMPRESSION_H264_LOOP = 19, H_COMPRESSION_H264_R

GB_LOOP = 20, H_COMPRESSION_HEVC = 21,
H_COMPRESSION_MPEG4 = 22 }

Sequence file compression algorithm.

- enum eHDescriptionFormat { H_UNICODE_FORMAT = 0, H_MBCS_FORMAT = 1, H_BYTE_FORMAT = 2 }

  Character set used for the "Description" parameter in the sequence header.

- enum eHProtectionStatus { H_PROTECTION_NONE = 0, H_PROTECTION_VALID, H_PROTECTION_I↩NVALID }

  Authorization code validity.

- enum HTimeSourceSyncStatus {
  H_SYNC_STATUS_UNSUPPORTED = 0, H_SYNC_STATUS_UNKOWN, H_SYNC_STATUS_FREERUN,
  H_SYNC_STATUS_SYNCHRONIZED,
  H_SYNC_STATUS_HOLDOVER }

  Sync status code for time source.

- enum eHBayerSampleFactor { H_BAYER_SAMPLEFACTOR_1 = 1, H_BAYER_SAMPLEFACTOR_2 = 2, H_BAYER_SAMPLEFACTOR_4 = 4, H_BAYER_SAMPLEFACTOR_8 = 8 }

  Sample factor used for the Bayer conversion.

- enum eHBayerPatternOrigin { H_BAYER_PATTERNORIGIN_GB = 0, H_BAYER_PATTERNORIGIN_GR, H_BAYER_PATTERNORIGIN_B, H_BAYER_PATTERNORIGIN_R }

  Pattern origin used for the Bayer conversion.

- enum eHBayerCalculationMode {
  H_BAYER_CALCMODE_FAST = 0, **H_BAYER_CALCMODE_ACCURATE**, **H_BAYER_CALCMODE_L↩APLACIAN1**, **H_BAYER_CALCMODE_LAPLACIAN2**,
  H_BAYER_CALCMODE_IPP, **H_BAYER_CALCMODE_IPP_DEMOSAIC_AHD** }

  Algorithm used for the Bayer conversion.

- enum eHColorBalanceAlgorithm {
  H_COLBALALGO_MEAN_EQUALIZATION = 0, **H_COLBALALGO_GRAY_WORLD**, **H_COLBALALG↩O_STANDARD_DEVIATION**, **H_COLBALALGO_MEAN_AND_STANDARD_DEVIATION**,
  H_COLBALALGO_IRIDAS_1D, **H_COLBALALGO_IRIDAS_3D_FAST**, **H_COLBALALGO_IRIDAS_3D↩_NORMAL**, **H_COLBALALGO_PRE_DEFINED** }

  Algorithm used for the color balance.

- enum eHAutoWhiteBalanceAlgorithm {
  H_AWB_MEAN_EQUALIZATION = 0, **H_AWB_GRAY_WORLD**, **H_AWB_STANDARD_DEVIATION**, **H↩_AWB_MEAN_AND_STANDARD_DEVIATION**,
  H_AWB_WHITE_WORLD, **H_AWB_SCALE_BY_MAX**, **H_AWB_PRE_DEFINED** }

  Algorithm used for the automatic color balance (white balancing).

- enum eHInputEvent { H_INPUT_EVENT_FALLING_EDGE = 0, H_INPUT_EVENT_RISING_EDGE }

  Input signal edges.

- enum eHInputLevel { H_INPUT_LEVEL_LOW = 0, H_INPUT_LEVEL_HIGH, H_INPUT_LEVEL_UNDEFI↩NED }

  Input signal levels.

- enum eHOutputLevel { H_OUTPUT_LEVEL_LOW = 0, H_OUTPUT_LEVEL_HIGH, H_OUTPUT_LEVEL_↩UNDEFINED }

  Output signal levels.

- enum eHVideoCodec {
  H_CODEC_UNKNOWN = 0, **H_CODEC_CINEFORM_RAW**, **H_CODEC_CINEFORM_HD**, **H_CODEC_↩H264**,
  H_CODEC_NPXJ, **H_CODEC_NPXH**, **H_CODEC_CUDA_NPXJ**, **H_CODEC_CUDA_NPXH**,
  H_CODEC_MJPEG, **H_CODEC_NPXHEVC**, **H_CODEC_NPXMPEG4**, **H_CODEC_CUDA_NPXHEVC**,
  H_CODEC_AMD_H264 }

  1st-party video codecs.

- enum eHSpecialCopyFormat { **H_SPECIAL_COPY_NONE** = 0, H_SPECIAL_COPY_JPEG, H_SPECIAL_COPY_YUV422, H_SPECIAL_COPY_UVY422, H_SPECIAL_COPY_YUV411, H_SPECIAL_COPY_YUV444, H_SPECIAL_COPY_RG30, H_SPECIAL_COPY_RG48, H_SPECIAL_COPY_BYR1, H_SPECIAL_COPY_BYR2, H_SPECIAL_COPY_BYR4, H_SPECIAL_COPY_Y16, H_SPECIAL_COPY_Y8, H_SPECIAL_COPY_BGR555, H_SPECIAL_COPY_BGR565, H_SPECIAL_COPY_BGRA, H_SPECIAL_COPY_H264 }

  Native support image format by AVI codec.
- enum eAdvanceSettingsLevel { **H_ADVANCE_LEVEL_BEGINNER** = 0, **H_ADVANCE_LEVEL_EXPERT** = 1, **H_ADVANCE_LEVEL_GURU** = 2 }

  The level determine how many settings are shown in the Advanced Settings dialog.
- enum eAdvanceSettingsFeatureType { H_ADVANCE_ENUMERATION = 0, H_ADVANCE_INTEGER_VALUE, H_ADVANCE_DOUBLE_VALUE, H_ADVANCE_BOOL_VALUE, H_ADVANCE_COMMAND, H_ADVANCE_OTHERS = 100 }

  AdvanceSettings support data type.
- enum eFastColorCorrectionMode { H_FAST_CORRECTION_NONE = 0, H_FAST_CORRECTION_MANUAL, H_FAST_CORRECTION_AUTO }

  Color correction mode.
- enum eHImageDateTimeFormat { H_DATETIME_STANDARD = 0, H_DATETIME_FULL, H_DATETIME_TIME_T, H_DATETIME_LTC }

  TIFF files DATETIME formats.
- enum eHMovieMode { H_MOVIE_UNDEFINED = 0, H_MOVIE_RECORDING, H_MOVIE_PLAYBACK }

  The current mode of a movie file.
- enum eHAviPlaybackFormat { H_AVI_PLAYBACK_NATIVE = 0, H_AVI_PLAYBACK_BGR, H_AVI_PLAYBACK_BGRX }

  Format of images read from an AVI.
- enum eHRotationInterpolationMode { H_ROTATION_INTER_NN = 0, H_ROTATION_INTER_LINEAR, H_ROTATION_INTER_CUBIC }

  Rotation interpolation mode.
- enum eHResizeInterpolationMode { H_RESIZE_INTER_NN = 0, H_RESIZE_INTER_LINEAR, H_RESIZE_INTER_CUBIC, H_RESIZE_INTER_LANCZOS, H_RESIZE_INTER_SUPER }

  Resize interpolation mode.
- enum eHMotionDetectionMode { H_MOTION_DETECTION_MEAN = 0, H_MOTION_DETECTION_ABS_DIFF, H_MOTION_DETECTION_MAX_ABS_DIFF, H_MOTION_DETECTION_EDGE, H_MOTION_DETECTION_EDGE_ADAPTIVE }

  Motion detection algorithm type.
- enum eHSobelMode { H_SOBEL_VERTICAL = 0, H_SOBEL_HORIZONTAL, H_SOBEL_BOTH }

  Sobel edge detection type.
- enum eMonoToPseudoColorLUT { H_MONO_TO_PSEUDO_COLOR_DISABLED = -1, H_MONO_TO_PSEUDO_COLOR_LUT_RAINBOW, H_MONO_TO_PSEUDO_COLOR_LUT_INVERTED_RAINBOW, H_MONO_TO_PSEUDO_COLOR_LUT_HOT, H_MONO_TO_PSEUDO_COLOR_LUT_COLD, H_MONO_TO_PSEUDO_COLOR_LUT_SATURATION_3, H_MONO_TO_PSEUDO_COLOR_LUT_SATURATION_5, H_MONO_TO_PSEUDO_COLOR_LUT_NEGATIVE }

  Pseudo color processing algorithm mode.

- enum eHGRABBER_NOTIFICATION { H_GRABBER_NOTIFICATION_CONNECTION_LOST = 0, H_GRABBER_NOTIFICATION_INVALID_SIGNAL, H_GRABBER_NOTIFICATION_SIGNAL_PRESENT, H_GRABBER_NOTIFICATION_CONNECTED, H_GRABBER_NOTIFICATION_DISCONNECTED, H_GRABBER_NOTIFICATION_CAPTURE_ERROR, H_GRABBER_NOTIFICATION_CAPTURE_EVENT_RECEIVED, H_GRABBER_NOTIFICATION_TIMESTAMP_SET, H_GRABBER_NOTIFICATION_FRAME_DROP, H_GRABBER_NOTIFICATION_PTP_CLOCK_STATUS }

  Notifications from the grabber.
- enum eHSUGGESTED_ACTION { H_SUGGESTED_ACTION_NONE = 0, H_SUGGESTED_ACTION_RELOAD_GRABBER, H_SUGGESTED_ACTION_UNLOAD_GRABBER, H_SUGGESTED_ACTION_PAUSE_STREAM, H_SUGGESTED_ACTION_RESUME_STREAM, H_SUGGESTED_ACTION_RESTART_STREAM, H_SUGGESTED_ACTION_OPEN_HARDWARESETTINGDLG, H_SUGGESTED_ACTION_ENABLE_FRAME_DROP, H_SUGGESTED_ACTION_DISABLE_FRAME_DROP }

  Actions suggested by the grabber along with a notification.
- enum eHMirror_Mode { H_MIRROR_HORZ = 0, H_MIRROR_VERT, H_MIRROR_BOTH }

  Mirroring mode.
- enum eHMerge_Mode { H_MERGE_HORZ = 0, H_MERGE_VERT }

  Image merging mode.
- enum eAudioFileFormat { H_AUDIO_FILE_AUD_PCM = 0, H_AUDIO_FILE_BWAVE_PCM }

  Support audio file format.
- enum eAudFileMode { H_AUD_FILE_AUD_BLOCK = 0, H_AUD_FILE_BWAVE_CONTINOUS }

  AUD file data type support.
- enum eHAudioFormat { H_AUDIO_UNKNOWN = 0, H_AUDIO_INT_MSB_PLANAR_PCM = 100, H_AUDIO_INT_PLANAR_PCM = 110, H_AUDIO_FLOAT_MSB_PLANAR_PCM = 120, H_AUDIO_FLOAT_PLANAR_PCM = 130, H_AUDIO_UINT_PLANAR_PCM = 180, H_AUDIO_INT_PACKED_PCM = 140, H_AUDIO_UINT_PACKED_PCM = 150, H_AUDIO_FLOAT_PACKED_PCM = 160, H_AUDIO_INT_MSB_PACKED_PCM = 170, H_AUDIO_FLOAT_MSB_PACKED_PCM = 190 }

  Audio format.
- enum eHAudioCompression { H_AUDIO_COMPRESSION_NONE = 0, H_AUDIO_COMPRESSION_MP3 = 1, H_AUDIO_COMPRESSION_AAC = 2 }

  Compressed audio format.
- enum eHAudioPlayEventCode { H_AUDIO_READ_FAILED = 0, H_AUDIO_UNSUPPORTED_FORMAT, H_AUDIO_SAMPLE_BUFFER_CHANGED }

  Event code for audio playback.
- enum eHAudioFileFormat { **H_AUDIO_FILE_NONE**, **H_AUDIO_FILE_AUD**, **H_AUDIO_FILE_BWAVE**, **H_AUDIO_FILE_MP3**, **H_AUDIO_FILE_AAC** }

  Audio file formats.
- enum eHExportFormat { H_EXPORT_BMP = 0, H_EXPORT_JPG, H_EXPORT_TIFF, H_EXPORT_PNG, H_EXPORT_FIT, H_EXPORT_JP2, H_EXPORT_DPX, H_EXPORT_AVI, H_EXPORT_SEQ, H_EXPORT_MOV }

  Linescan manager exported file format.
- enum eDAQDeviceParamType { H_DAQ_PARAM_ENUM, H_DAQ_PARAM_FLOAT, H_DAQ_PARAM_INT }

  DAQ device parameter data type.
- enum eTimeSourceParamType { H_TS_PARAM_ENUM, H_TS_PARAM_FLOAT, H_TS_PARAM_INT }

  Time source device parameter data type.
- enum eHCayer_Compression { H_CYR_VANILLA, H_CYR_VANILLA_LOSSY_LOW, H_CYR_VANILLA_LOSSY_MED, H_CYR_VANILLA_LOSSY_HI }

Cayer compression type.

- enum eHMetadataFormat {
  H_METADATA_UNKNOWN = 0, H_METADATA_BOOL, H_METADATA_BYTE, H_METADATA_SHORT, H_METADATA_USHORT, H_METADATA_INT, H_METADATA_UINT, H_METADATA_DOUBLE, H_METADATA_STRING, H_METADATA_BINARY, H_METADATA_INT64, H_METADATA_UINT64 }

  Metadata content format.

- enum eDeviceAcclType {
  H_GPU_DEVICE_UNKNOWN, H_CPU_DEVICE_ONLY, H_GPU_DEVICE_NVIDIA_CUDA, H_GPU_DEVICE_NVIDIA_OPENCL,
  H_GPU_DEVICE_AMD, H_GPU_DEVICE_INTEL }

  Supported GPU type.

- enum eJpegEncodeAlgoType { H_JPEG_EN_ALGO_IPP, H_JPEG_EN_ALGO_GPU, H_JPEG_EN_ALGO_TURBO }

  JPEG encoding type.

- enum eJpegImageExportType { H_JPEG_IMAGE_EXPORT_IPP, H_JPEG_IMAGE_EXPORT_GPU, H_JPEG_IMAGE_EXPORT_TURBO, H_JPEG_IMAGE_EXPORT_FREE_IMAGE }

  JPEG exporting type.

- enum eJpegDecodeAlgoType { H_JPEG_DE_ALGO_IPP, H_JPEG_DE_ALGO_TURBO, H_JPEG_DE_ALGO_GPU_INTEL }

  JPEG decoding type.

- enum eHErrorCode {
  H_ERROR_SUCCESS = 0, H_ERROR_NOAUTHORIZATION, H_ERROR_FAILEDREGISTER, H_ERROR_EXPORT_IN_PROGRESS,
  H_ERROR_INVALID_COMPRESSION, H_ERROR_NO_BEGINEXPORT, H_ERROR_NO_IMAGE_DATA, H_ERROR_INVALID_DIGIT,
  H_ERROR_NO_GRABMAN, H_ERROR_INI_FAIL, H_ERROR_CREATE_BUFFER, H_ERROR_BUFFER_INTERNAL,
  H_ERROR_NO_GRABBER, H_ERROR_CONVERT_BGR24, H_ERROR_CONVERT_BGRX, H_ERROR_CONVERT_UNSUPPORTED,
  H_ERROR_CONVERT_YUV422, H_ERROR_CONVERT_UVY422, H_ERROR_CONVERT_UVY411, H_ERROR_CONVERT_UVY444,
  H_ERROR_CREATE_SYSTEM_ENUM, H_ERROR_CREATE_CLASS_ENUM, H_ERROR_MONIKER_NULL, H_ERROR_LOADING_VIDEO_CODEC,
  H_ERROR_LOADING_AUDIO_SOURCE, H_ERROR_LOADING_AUDIO_CODEC, H_ERROR_NO_CODEC_SETTINGS, H_ERROR_CREATE_GRAPH_BUILDER,
  H_ERROR_CREATE_CAPTURE_BUILDER, H_ERROR_CREATE_FEEDER, H_ERROR_ADD_FEEDER, H_ERROR_ADD_VCODEC,
  H_ERROR_ADD_ACODEC, H_ERROR_ADD_AUDIO_SOURCE, H_ERROR_ADD_MUX_FILE, H_ERROR_CONNECT_VSOURCE_VCODEC,
  H_ERROR_CONNECT_VCODEC_MUX, H_ERROR_CONNECT_VSOURCE_MUX, H_ERROR_CONNECT_ASOURCE_ACODEC, H_ERROR_CONNECT_ACODEC_MUX,
  H_ERROR_CONNECT_ASOURCE_MUX, H_ERROR_CONNECT_SOURCE_SPLITTER, H_ERROR_CONNECT_SPLITTER_RENDERER, H_ERROR_SYNCH_SOURCE,
  H_ERROR_INTERLEAVE_MODE, H_ERROR_INTERLEAVE_TIME, H_ERROR_SEQ_CANTCREATEFILE, H_ERROR_NOSEQ,
  H_ERROR_WRITE_SEQ, H_ERROR_SEQ_INVALIDMAGICNUMBER, H_ERROR_SEQ_WRONGHEADERSIZE, H_ERROR_SEQ_INVALIDLENGTH,
  H_ERROR_SEQ_INVALIDFRAMERATE, H_ERROR_SEQ_INVALIDDATA, H_ERROR_SEQ_INVALIDWIDTH, H_ERROR_SEQ_INVALIDHEIGHT,
  H_ERROR_SEQ_INVALIDDEPTH, H_ERROR_SEQ_INVALIDIMAGESIZE, H_ERROR_SEQ_UNSUPPORTED_FORMAT, H_ERROR_SECTORBYTES_ZERO,
  H_ERROR_FILENOTFOUND, H_ERROR_CANTCREATEFILE, H_ERROR_INVALID_INDEX, H_ERROR_OUTOFMEMORY,
  H_ERROR_IMAGEFORMATCHANGED, H_ERROR_TOOMUCHDATA, H_ERROR_GDIDRAW, H_ERROR_SCRIPTRUNNING,
  H_ERROR_IMAGE_FORMAT, H_ERROR_SCRIPTER_NOT_RUNNING, H_ERROR_STOP_PLAYBACK,

H_ERROR_STREAMING,
H_ERROR_DONTSUPPORTREWRITE, H_ERROR_UNSUPPORTED, H_ERROR_UNSUPPORTED_S↩
ETTINGS, H_ERROR_UNSUPPORTED_ADJUSTMENTS,
H_ERROR_NOCOMPORT, H_ERROR_EXPORT_FAILED, H_ERROR_INPUTSUPPORT_LOAD, H_ER↩
ROR_OUTPUTSUPPORT_LOAD,
H_ERROR_VIDEOOUTPUTSUPPORT_LOAD, **H_ERROR_H264_ENCODE**, **H_ERROR_H264_DECODE**,
H_ERROR_UNSUPPORTED_ADVANCEFEATURE,
H_ERROR_NOBITMAP, H_ERROR_QUICKTIME_INIFAIL, H_ERROR_WRONG_MODE, H_ERROR_Q↩
UICKTIME_ERROR,
H_ERROR_ROTATION_ERROR, H_ERROR_MIRROR_ERROR, H_ERROR_RESIZE_ERROR, **H_ERR↩
OR_MOTION_ERROR**,
H_ERROR_MERGE_ERROR, **H_ERROR_PARAM_NOT_SUPPORTED**, H_ERROR_INPUTSUPPORT_↩
DEV_INIT_FAILED, H_ERROR_INPUTSUPPORT_DEV_UNINIT_FAILED,
H_ERROR_INPUTSUPPORT_MONITOR, H_ERROR_OUTPUTSUPPORT_DEV_INIT_FAILED, H_ERR↩
OR_OUTPUTSUPPORT_DEV_UNINIT_FAILED, H_ERROR_ADD_FILEREADER,
H_ERROR_ADD_AVISPLITTER, H_ERROR_CREATE_RENDER, **H_ERROR_PULSEGENSUPPORT_L↩
OAD**, **H_ERROR_PULSEGENSUPPORT_DEV_INIT_FAILED**,
H_ERROR_PULSEGENSUPPORT_DEV_UNINIT_FAILED, **H_ERROR_PULSEGENSUPPORT_START↩
_FAILED**, **H_ERROR_PULSEGENSUPPORT_STOP_FAILED**, H_ERROR_NO_PROCESSING_APPLIED,
H_ERROR_COLOR_BALANCING, H_ERROR_BAYER_CONVERSION, H_ERROR_DAQSUPPORT_L↩
OAD, **H_ERROR_DAQSUPPORT_INIT_FAILED**,
H_ERROR_DAQSUPPORT_UNINIT_FAILED, H_ERROR_DAQSUPPORT_READ_FAILED, **H_ERROR↩
_DAQSUPPORT_SEEK_FAILED**, H_ERROR_DAQSUPPORT_CANTCREATEFILE,
H_ERROR_DAQSUPPORT_INVALID_INDEX, H_ERROR_DAQSUPPORT_NODAQ, H_ERROR_DAQS↩
UPPORT_WRITE_DAQ, H_ERROR_DAQSUPPORT_FORMAT_CHANGED,
H_ERROR_AUDIO_FORMAT, H_ERROR_AUDIO_LOAD, H_ERROR_AUDIO_INPUTSUPPORT_LOAD,
H_ERROR_AUDIO_OUTPUTSUPPORT_LOAD,
H_ERROR_AUDIO_INPUTSUPPORT, H_ERROR_AUDIO_OUTPUTSUPPORT, H_ERROR_AUDIO_CA↩
NTCREATEFILE, H_ERROR_AUDIO_NOAUD,
H_ERROR_AUDIO_WRITE_AUD, H_ERROR_AUDIO_FORMAT_CHANGED, H_ERROR_AUDIO_INVA↩
LID_INDEX, H_ERROR_AUDIO_INVALID_FORMAT,
H_ERROR_AUDIO_CHANNEL_MASK_MATCH, H_ERROR_AUDIO_READ_FAILED, H_ERROR_CANT↩
_ADD_FRAME, H_ERROR_UNSUPPORT_FUNCTION,
H_ERROR_CREATE_SOCKET, H_ERROR_BIND_SOCKET, H_ERROR_LISTEN_SOCKET, H_ERROR↩
_SHUTDOWN_SOCKET,
H_ERROR_CLOSE_SOCKET, H_ERROR_INVALID_SOCKET, H_ERROR_SERVER_NOT_RUNNING,
H_ERROR_SERVER_ALREADY_OPEN,
H_ERROR_CLIENT_ALREADY_CONNECTED, H_ERROR_CONNECTION_FAILED, H_ERROR_SEND,
H_ERROR_OUT_OF_RANGE,
H_ERROR_NO_FILE, H_ERROR_H264_UNSUPPOT_IMAGE_SIZE, H_ERROR_H264_ENCODING, H_↩
ERROR_H264_DECODING,
H_ERROR_CREATE_SAMPLE_GRABBER, H_ERROR_SAMPLE_GRABBER_CALLBACK, H_ERROR_↩
ADD_SAMPLE_GRABBER, H_ERROR_CONNECT_ASOURCE_ALEVEL,
H_ERROR_CONNECT_ALEVEL_ACODEC, H_ERROR_CONNECT_ALEVEL_MUX, H_ERROR_NO_A↩
UDIO_SOURCE, H_ERROR_NO_RECORDING,
H_ERROR_NO_CAPTURE_GRAPH, H_ERROR_FAILED_TO_ADD_TAG, H_ERROR_FRAME_SKIPPED,
H_ERROR_AVI_TAG_NOT_FOUND,
H_ERROR_REGISTRY_READ, H_ERROR_REGISTRY_WRITE, H_ERROR_CREATEC_UNSUPPORT↩
_FORMAT, H_ERROR_INVALID_LUT_FILE,
H_ERROR_NOT_CONNECTED, H_ERROR_SOCKET_ALREADY_CREATED, H_ERROR_WRITE_ER↩
ROR, H_ERROR_CREATE_FILE_ERROR,
H_ERROR_MALFORMED_XML_FILE, H_ERROR_MP4_VIDEO_READ, H_ERROR_MP4_AUDIO_READ,
H_ERROR_MPEG4_ENCODING,
H_ERROR_MPEG4_DECODING, H_ERROR_HEVC_ENCODING, H_ERROR_HEVC_DECODING, H_E↩
RROR_DECODE_MORE_DATA,
H_ERROR_AUIDO_PROCESS_MORE_DATA, H_ERROR_AUIDO_PROCESS_NO_DATA, H_ERROR_↩
AAC_DECODING, H_ERROR_MP3_DECODING,
H_ERROR_UNSUPPORTED_INTERFACE, H_ERROR_GPU_IMAGE_PUSH }

Hermes error codes.

4.1.1 Enumeration Type Documentation

4.1.1.1 enum eHSupportedModule

The supported cameras & frame grabbers.

Enumerator

H_ABSOFTWARE_ACTIVE_GIGE NpxActiveGigE.dll (A&B Software GigE Vision cameras)

H_ABSOFTWARE_ACTIVE_USB NpxActiveUSB.dll (A&B Software USB cameras)

H_ACTIVESILICON_FIREBIRD NpxASFirebird.dll (Active Silicon Firebird)

H_ACTIVESILICON_PHOENIX NpxPhoenix.dll (Active Silicon Phoenix)

H_AJA_KONA NpxKona.dll (AJA Kona serie)

H_ANDOR_LUCA NpxAndor.dll (Andor CCD)

H_ANDOR_NEO NpxAndorNeo.dll (Andor sCMOS (Neo/Zyla))

H_AVT_DCAM NpxAVT.dll (AVT DCAM FireGrab)

H_AVT_DCAM_UNI NpxAVTUNI.dll (AVT DCAM Uniform API)

H_AVT_UNIVERSAL NpxAVTUniversal.dll (AVT Universal API (AVT + Prosilica))

H_AXIS NpxAxis.dll (Axis camera)

H_BASLER_IEEE1394 NpxBasler1394B.DLL (Basler Pylon IEEE1394 cameras)

H_BASLER_GIGE NpxBaslerGigE.DLL (Basler Pylon GigE cameras)

H_BASLER_BCAM NpxBasler.dll (Basler BCAM cameras (obsolete, use the pylondriver))

H_BASLER_PYLON NpxBaslerPylon.dll (Basler cameras (GigE, firewire, USB3))

H_BASLER_TOF NpxBaslerToF.dll (Basler ToF cameras)

H_BAUMER_FW NpxBaumerFW.dll (Baumer Firewire Camera)

H_BAUMER2 NpxBaumer2.dll (Baumer Firewire Camera version2)

H_BITFLOW_CL NpxBitFlow.dll (BitFlow CameraLink & Analog)

H_BITFLOW_CXP NpxBitFlowCXP.dll (BitFlow CoaXPress)

H_BLACK_MAGIC NpxBlackMagic.dll (Black Magic Design)

H_CCAM NpxCCam.dll (C-CAM Tehnologies USB2 camera)

H_CCDDIRECT_MICROPIX NpxMicroPix.dll (CCD Direct Micropix)

H_CMU_IEEE1394 NpxIEEE1394.dll (DCAM Camera, Sony IEEE1394 camera)

H_COOKE_PCO NpxPco.dll (Cooke PCO)

H_COOKE_PIXELFLY NpxPixelFly.dll (Cooke PixelFly)

H_COOKE_SENSICAM NpxSensiCam.dll (Cooke Sensicam)

H_COOKE_SENSICAM_2 NpxSensicam2.dll (Cooke Sensicam (the new driver))

H_CORECO_BANDIT2CV NpxBanditIICV.dll (Dalsa Bandit 2 CV)

H_CORECO_BANDIT2MV NpxBanditIIMV.dll (Dalsa Bandit 2 MV)

H_CORECO_BANDIT2RGB NpxBanditIIRGB.dll (Dalsa Bandit 2 RGB)

H_CORECO_GIGE NpxCorecoGigE.dll (Dalsa GigE Camera)

H_CORECO_VIPERDIG NpxViperDig.dll (Dalsa Viper Digital)

H_CORECO_PCCAMLINK NpxPcCamLink.dll (Dalsa PC-CAM link)

H_CORECO_PC2CAMLINK NpxPc2CamLink.dll (Dalsa PC2-CAM link)

H_CORECO_PC2COMP NpxPc2Comp.dll (Dalsa PC2-COMP)

H_CORECO_PC2VISION NpxPC2Vision.dll (Dalsa PC2-Vision)

H_CORECO_X64 NpxCorecoX64.dll (Dalsa X-64)

H_DATAPATH_VISIONRGB NpxVisionRGB.dll (DataPath Vision RGB)

H_EMERGENT NpxEmergent.dll (Emergent Vision Technologies)

H_EMS_PHYNXRGB NpxPhynxRGB.dll (EMS Phynx RGB)

H_EMS_XTREMERGB NpxVisionRGB.dll (EMS Xtreme RGB)

H_EMVACAM NpxEmvaCam.dll (Emva GeniCam)

H_EPIPHAN NpxEpiphan.dll (Epiphan)

H_EPIX_D2X NpxEpixD2X.dll (Epix Digital family)

H_EPIX_RAPTOR NpxEpixRaptor.dll (Epix - Raptor Photonics)

H_EPIX_SV5 NpxEpixSV5.dll (Epix SV5)

H_EURESYS_GRABLINKVALUE NpxGrabLink.dll (Euresys Grablink Series)

H_EURESYS_IOTA NpxIota.dll (Euresys Domino Series)

H_EURESYS_PICOLO NpxPicolo.dll (Euresys Picolo Series)

H_EURESYS_PICOLO_HD NpxPicoloHD.dll (Euresys Picolo HD Series)

H_PICOLO_H264_V16 NpxPicoloV16.dll (Euresys Picolo H264 V16)

H_PICOLO_H264_UX NpxPicoloUx.dll (Euresys Picolo H264 Ux)

H_FCAV_ARINC818 NpxFCAV.dll (FCAV ARINC 818 Family)

H_FOCULUS_CAMERA NpxFOculus.dll (Foculus Firewire Camera)

H_FORESIGHTIMAGING_ACCUSTREAM NpxAccuStream.dll (Foresight Imaging AccuStream)

H_FR_STREAM_USB NpxFRStreamUSB.dll (FR-Stream USB Camera) (actually a custom copy of Ximea)

H_GEVICAM NpxGEViCAM.dll (GEViCAM Camera Family)

H_GIGALINX NpxGigaLinx.dll (GigaLinx Camera Family)

H_GIGALINX_SONY NpxGigaLinx.dll (Sony GigE Vision)

H_GRT_MATRIX NpxGRTMatrix.dll (Great River Matrix Family)

H_GRT_MATRIX_HL2V NpxGRTMatrixHL2V.dll (Great River Matrix Hotlink Family)

H_HAMAMATSU_DCAM NpxHamamatsu.dll (Hamamatsu DCAM)

H_HASOTEC_FG30 NpxHaSoTec.dll (Hasotec FG30)

H_HITACHI_DCAM NpxHitachiDCAM.dll (Hitachi DCAM Family)

H_HL2V NpxHL2v.dll (Hotlink Card)

H_IDS_UEYE NpxIDSUEye.dll (IDS U-Eye Family)

H_IMAGINGSOURCE NpxImagingSource.dll (ImagingSource GigE camera)

H_IMITECH_1394 NpxIMI1394.dll (Imi Tech 1394 camera)

H_IMITECH_GIGE NpxIMIGigE.dll (Imi Tech GigE camera)

H_IMITECH_NEPTUNE NpxIMINeptune.dll (Imi Tech Neptune driver)

H_IMPERX_BOBCAT_GIGE NpxGEViCAM.dll (ImperX Bobcat GigE)

H_IMPERX_FRAMELINK NpxFrameLink.dll (ImperX Frame Link)

H_IMPERX_FRAMELINK_EXPRESS NpxImperxLink.dll (ImperX Frame Link Express)

H_IMPERX_VCE_EXPRESS NpxVCEExpress.dll (ImperX VCE Express)

H_IMPERX_SDI_EXPRESS NpxSDIExpress.dll (ImperX HD SDI Express)

H_IMPERX_GIGE NpxGigE.dll (ImperX GigE)

H_IMPERX_VCEPRO NpxVCEPro.dll (ImperX VCE Pro)

H_INTEGRAL_FLASHBUSDX NpxFlashBusDX.dll (Integral Technology Flashbus DX Family)

H_INTEGRAL_FLASHBUSMV NpxFlashbus.dll (Integral Technology Flashbus MV Family)

H_INTEGRAL_FLASHBUSMX NpxFlashBusMX.dll (Integral Technology Flashbus MX Family)

H_INTEGRAL_SPECTRIM NpxSpectrim.dll (Integral Technology Spectrim)

H_IP_CAMERA NpxIPCamera.dll (IP Web compatible Camera)

H_ISG_1394 NpxISG1394.dll (Imaging Solutions Group 1394 cameras)

H_JAIGENICAM_GIGE NpxJAI.dll (Jai GeniCam GigE)

H_JAIGENICAM2 NpxJAI2.dll (Jai GeniCam GigE & USB3)

H_JAIPULNIX_GIGE NpxGigE.dll (Jai/Pulnix GigE)

H_JENOPTIK_DIJ NpxJenoptikDij.dll (Jenoptik Progres GRYPHAX, C14Plus, CFcool, MFCool)

H_JENOPTIK_PROGRESC10 NpxProgResC10.dll (Jenoptik ProgRes C10)

H_JENOPTIK_PROGRESCXX NpxProgResXX.dll (Jenoptik ProgRes C10/C12/C14/M14 Plus)

H_JENOPTIK_PROGRESC14 NpxProgResC14.dll (Jenoptik ProgRes C14)

H_KAPPA_FW NpxKappaFW.dll (Kappa Firewire Camera)

H_KAPPA_HIRES3XR NpxKappaHires3XR.dll (Kappa Hires3XR (GigE) Camera)

H_KAYA NpxKaya.dll (KAYA frame grabber)

H_KINECT NpxKinect.dll (Kinect for Windows)

H_LEUTRON_PICSIGHT_GIGE NpxPicSight.dll (Leutron PicSight GigE)

H_LEUTRON_SIMPLON_GIGE NpxLeutronSimplon.dll (Leutron Simplon GigE)

H_LUMENERA_GIGE NpxLumenera.dll (Lumenera (GigE family))

H_LUMENERA_USB2 NpxLumenera.dll (Lumenera (USB 2.0 family))

H_MATRIXVISION NpxMatrixVision.dll (Matrix Vision GigE and USB3 camera)

H_MATRIXVISION_BLUECOUGAR NpxBlueCOUGAR.dll (Matrix Vision BlueCOUGAR GigE camera)

H_MATRIXVISION_BLUEFOX NpxBlueFOX.dll (Matrix Vision BlueFOX USB camera)

H_MATRIXVISION_HYPERION NpxHyperion.dll (Matrix Vision Hyperion (Digital) Frame Grabber)

H_MATRIXVISION_SIGMA NpxSigma.dll (Matrix Vision Sigma (Analogue) Frame Grabber)

H_MATRIXVISION_TITAN NpxTitan.dll (Matrix Vision Titan (Analogue) Frame Grabber)

H_MATROX_CXP NpxMatroxCXP.dll (Matrox Radient CoaXPress)

H_MATROX_CL NpxMatroxCL.dll (Matrox Camera Link)

H_MATROX_HELIOS NpxHelios.dll (Matrox Helios)

H_MATROX_METEOR2_CL NpxMtxM2CL.dll (Matrox Meteor 2 Camera link)

H_MATROX_METEOR2_DIG NpxMtxM2Dig.dll (Matrox Meteor 2 Digital)

H_MATROX_METEOR2_MC NpxMtxM2MC.dll (Matrox Meteor 2 Multi-Channel)

H_MATROX_METEOR2_STD NpxMtxM2STD.dll (Matrox Meteor 2 Standard)

H_MATROX_MORPHIS NpxMtxMorphis.dll (Matrox Morphis)

H_MATROX_RADIENT NpxRadient.dll (Matrox Radient eCL)

H_MATROX_SOLIOS_XCL NpxSolios.dll (Matrox Solios XCL)

H_MATROX_ODYSSEY NpxOdyssey.dll (Matrox Odyssey)

H_MATROX_ORION NpxOrion.dll (Matrox Orion)

H_MATROX_VIO_XCL NpxVIO.dll (Matrox VIO XCL)

H_MIGHTEX NpxMightex.dll (Mightex CMOS buffered camera)

H_MIGHTEX_CCD NpxMightexCCD.dll (Mightex CCD area camera)

H_NET_GIGE NpxNETGigE.dll (NET GigE cameras)

H_NET_ICUBE NpxICube.dll (NET iCube USB cameras)

H_NI_GRABBERS_0 NpxNI1429.dll (National Instruments Frame Grabbers)

H_NIKON_D7000 NpxNikonD7000.dll (Nikon D7000 Camera)

H_NORPIX_VIRTUAL NpxVirtual.dll (Norpix Virtual Grabber)

H_OLYMPUS_DP70 NpxDP70.dll (Olympus DP-70)

H_ONVIF NpxONVIF.DLL (Onvif standard cameras)

H_OPTRONICS_MACROFIRE NpxMacroFire.dll (Optronics MacroFire)

H_OPTRONICS_MAGNAFIRE NpxMagnafireSP.dll (Optronics Magnafire SP)

H_OPTRONICS_MICROFIRE NpxMicroFire.dll (Optronics Microfire)

H_PIXELINK_PLA500 NpxPLA500.dll (Pixelink PL-A544 analog converter)

H_PIXELINK_PLA600 NpxPLA600.dll (Pixelink PL-A6XX family)

H_PIXELINK_PLA700 NpxPLA700.dll (Pixelink PL-A7XX family)

H_PIXELINK_TM1020 NpxTM1020.dll (Pixelink TM-1020 camera)

H_PIXELINK_VCA1281 NpxVCA1281.dll (Pixelink VCA-1281 camera)

H_PIXERA_PENGUIN NpxPenguin600CL.dll (Pixera Penguin Family)

H_PLEORA_EBUS NpxPleoraEbus.dll (Pleora GigE and USB3)

H_PLEORA_PT1000CL NpxPleora.dll (Pleora PT1000-CL (& other iPORT))

H_PRINCETON NpxPrinceton.dll (Princeton Instruments Cameras)

H_PROSILICA NpxProsilica.dll (Prosilica Firewire camera)

H_PROSILICA_GIGE NpxProsilicaGigE.dll (Prosilica GigE camera)

H_PTGREY NpxPTGrey.dll (Point Grey Research (GEV & firewire & USB))

H_PTGREY_LADYBUG NpxLadybug.dll (Point Grey Research LadyBug (USB3))

H_QIMAGING_MP NpxQImagingMP.dll (QImaging MicroPublisher)

H_QIMAGING_RETIGA NpxQImaging.dll (QImaging Retiga, QICAM & Rolera lines)

H_REDLAKE_MEGAPLUSII NpxMegaPlusII1394.dll (RedLake MegaPlus II 1394 camera)

H_ROPER_CASCADE NpxCoolSnapCF.dll (Roper Cascade)

H_ROPER_COOLSNAPCF NpxCoolSnapCF.dll (Roper CoolSnap CF)

H_SBF_FIREWIRE NpxSBFocal.dll (Santa Barbara Focal Plane (any))

H_SCION_CFW NpxScion.dll (Scion CFW 1312C/1310C/1308C/1312M/1310M/1308M)

H_SILICONSOFTWARE_MICROENABLE NpxMicroEnable.dll (Silicon Software MicroEnable (CL-Frame grabber))

H_SILICONSOFTWARE_MICROENABLE5 NpxMicroEnable5.dll (Silicon Software MicroEnable (CoaXPress grabber))

H_SILLICONIMAGING_GIGE NpxGigE.dll (Sillicon Imaging Gige)

H_SPINNAKER NpxSpinnaker.dll (Point Grey Flir Spinnaker, USB3 and GigE)

H_SPOT NpxSpot.dll (SPOT Insight $***$ to be changed to H_SPOT)

H_SUMIX_12A2 NpxSumix12A2.dll (Sumix 12A2 GigE Camera)

H_SUMIX_150 NpxSumix150.dll (Sumix 150 USB Camera)

H_SUMIX_15M5X NpxSumix15M5x.dll (Sumix 15M5x USB 3.0 cameras)

H_SUMIX_16EXX NpxSumix16eXX.dll (Sumix 16eXX USB 3.0 cameras)

H_SUMIX_M7X NpxSumixM7x.dll (Sumix M7X USB camera)

H_SUMIX_M8X NpxSumixM8x.dll (Sumix M8X USB camera)

H_SVS_VISTEK NpxSVSVistek.dll (SVS Vistek GigE & USB3 cameras)

H_SVS_VISTEK_GIGE NpxSVSGigE.dll (SVS Vistek GigE camera)

H_TATTILE_DUPLEX NpxTattileDuplex.dll (Tattile Duplex)

H_VIEWORKS NpxVieworks.dll (Vieworks cameras)

H_VIMBA NpxVimba.dll (AVT cameras)

H_VRMAGIC_RIC10 NpxVRmagicRIC10.dll (VRmagic RIC10 cameras)

H_WDM_CAMERA NpxWDMCamera.dll (DirectShow compatible capture device)

H_XCELERA NpxXcelera.dll (Dalsa Xcelera/X64)

H_XIMEA NpxXimea.dll (XIMEA USB Camera)

H_XENICS NpxXenics.dll (Xenics Camera)

4.1.1.2 enum eHGrabberInfoType

The type of grabber information.

Enumerator

H_GRABBER_INFO_TYPE_LONG Long.

H_GRABBER_INFO_TYPE_DOUBLE Double.

H_GRABBER_INFO_TYPE_STRING String.

4.1.1.3 enum eHJpegQuality

The JPEG quality. Higher quality means slower encoding and bigger files.

Enumerator

H_JPEG_DEFAULT Default compression algorythm.

H_JPEG_FAST Fast compression algorythm.

H_JPEG_ACCURATE Accurate compression algorythm.

H_JPEG_QUALITYSUPERB Superb quality, slowest encoding speed.

H_JPEG_QUALITYGOOD Good quality, slower encoding speed.

H_JPEG_QUALITYNORMAL Normal quality.

H_JPEG_QUALITYAVERAGE Average quality, fast encoding speed.

H_JPEG_QUALITYBAD Bad quality, fastest encoding speed.

4.1.1.4 enum eHImageFormat

The image format indicate how the pixel data is stored.

Enumerator

H_IMAGE_UNKNOWN Unknown format.

H_IMAGE_MONO Monochrome Image (LSB)

H_IMAGE_MONO_BAYER Raw Bayer Image (treated as H_IMAGE_MONO)

H_IMAGE_BGR BGR Color Image.

H_IMAGE_PLANAR Planar Color Image.

H_IMAGE_RGB RGB Color Image.

H_IMAGE_BGRx BGRx Color Image.

H_IMAGE_YUV422 YUV422.

H_IMAGE_YUV420_PLANAR YUV420 planar.

H_IMAGE_UVY422 UVY422.

H_IMAGE_UVY411 UVY411.

H_IMAGE_UVY444 UVY444.

H_IMAGE_BGR555_PACKED PhynxRGB format.

H_IMAGE_BGR565_PACKED PhynxRGB format.

H_IMAGE_MONO_MSB Only for $>$ 8 bit per pixel, MSB align litle endian 10 bit: JIHGFEDC BA000000.

H_IMAGE_MONO_BAYER_MSB Only for $>$ 8 bit per pixel, MSB align.

H_IMAGE_MONO_MSB_SWAP Only for $>$ 8 bit per pixel, MSB align big endian 10 bit: BA000000 JIHGF↩ EDC.

H_IMAGE_MONO_BAYER_MSB_SWAP Only for > 8 bit per pixel, MSB align.

H_IMAGE_BGR10_PPACKED Only for 10 bit per pixel, LSB align.

H_IMAGE_BGR10_PPACKED_PHOENIX Only for 10 bit per pixel, LSB align, RRRRRRRR RR00GGGG GGGGGGBB BBBBBBBB.

H_IMAGE_RGB10_PPACKED_PHOENIX Only for 10 bit per pixel, LSB align, BBBBBBBB BB00GGGG G↩ GGGGGRR RRRRRRRR.

H_IMAGE_MONO_PPACKED Only for > 8 bit per pixel, MSB align.

H_IMAGE_MONO_BAYER_PPACKED Only for > 8 bit per pixel, MSB align.

H_IMAGE_MONO_PPACKED_8448 Only for > 8 bit per pixel, MSB align.

H_IMAGE_MONO_BAYER_PPACKED_8448 Only for > 8 bit per pixel, MSB align.

H_IMAGE_GVSP_BGR10V1_PACKED Only for 10 bit per pixel(From Gige Vision)BBBBBBBB GGGGGG↩ GG RRRRRRRR 00BBGGRR.

H_IMAGE_GVSP_BGR10V2_PACKED Only for 10 bit per pixel(From Gige Vision)00BBBBBB BBGGGGGG GGGGRRRR RRRRRRRR.

H_IMAGE_MONO_PPACKED_8844 Only for > 8 bit per pixel, MSB align.

H_IMAGE_MONO_BAYER_PPACKED_8844 Only for > 8 bit per pixel, MSB align.

H_IMAGE_PHOTON_FOCUS_DR_MONO Photon Focus GigE double rate mono format.

H_IMAGE_PHOTON_FOCUS_DR_BAYER Photon Focus GigE double rate bayer format.

H_IMAGE_MONO80_64LSB_16MSB 8 pixels 10 bit (8 x 8 LSB bits + 8 x 2 MSB bits)

H_IMAGE_GVSP_PIX_MONO10_PACKED 4 pixels 10 bit packed DDDDDDDD DDCCCCCC CCCCBBBB BBBBBBAA AAAAAAAA

H_IMAGE_GVSP_PIX_MONO10_BAYER_PACKED 4 pixels 10 bit packed DDDDDDDD DDCCCCCC C↩ CCCBBBB BBBBBBAA AAAAAAAA

H_IMAGE_GRT_DUAL_IMAGE_RGB Great River custom format.

H_IMAGE_PHOTON_FOCUS_QR_MONO Photon Focus GigE quad rate mono format.

H_IMAGE_PHOTON_FOCUS_QR_BAYER Photon Focus GigE quad rate bayer format.

4.1.1.5 enum eHCompression

Sequence file compression algorithm.

Enumerator

H_COMPRESSION_NONE Uncompressed (raw) images.

H_COMPRESSION_JPEG JPEG compressed images.

H_COMPRESSION_RLE RLE compressed images (stp3 algo)

H_COMPRESSION_HUFFMAN HUFFMAN compressed images (stp3 algo)

H_COMPRESSION_LZ LZ compressed images (stp3 algo)

H_COMPRESSION_RLE_FAST RLE Fast compressed images (ippwrapper)

H_COMPRESSION_HUFFMAN_FAST HUFFMAN Fast compressed images (ippwrapper)

H_COMPRESSION_LZ_FAST LZ Fast compressed images (ippwrapper)

H_COMPRESSION_H264 H264 compressed images, BGR after uncompressed (default)

H_COMPRESSION_WAVELET Wavelet compressed images.

H_COMPRESSION_H264_RGB H264 compressed images, RGB after uncompressed (specific from some cameras)

H_COMPRESSION_JPEG_LOSSLESS Jpeg lossless jpeg compressed SEQ.

H_COMPRESSION_CAYER Createc cayer compression.

H_COMPRESSION_H264_LOOP Loop H264 compression,BGR when uncompressed (default)

H_COMPRESSION_H264_RGB_LOOP Loop H264 compression,RGB when uncompressed.

H_COMPRESSION_HEVC HEVC compression,BGR when uncompressed.

H_COMPRESSION_MPEG4 MPEG4 compression,BGR when uncompressed.

4.1.1.6 enum eHDescriptionFormat

Character set used for the "Description" parameter in the sequence header.

Enumerator

H_UNICODE_FORMAT Unicode string (default)
H_MBCS_FORMAT MBCS string.
H_BYTE_FORMAT User defined data (raw bytes)

4.1.1.7 enum eHProtectionStatus

Authorization code validity.

Enumerator

H_PROTECTION_NONE No code found.
H_PROTECTION_VALID Valid code.
H_PROTECTION_INVALID Invalid or expired code.

4.1.1.8 enum HTimeSourceSyncStatus

Sync status code for time source.

Enumerator

H_SYNC_STATUS_UNSUPPORTED Doesn't support sync status detection.
H_SYNC_STATUS_UNKOWN Unknown sync status.
H_SYNC_STATUS_FREERUN Free run mode.
H_SYNC_STATUS_SYNCHRONIZED Time source synchronized.
H_SYNC_STATUS_HOLDOVER Time source out of sync.

4.1.1.9 enum eHBayerSampleFactor

Sample factor used for the Bayer conversion.

Enumerator

H_BAYER_SAMPLEFACTOR_1 1:1
H_BAYER_SAMPLEFACTOR_2 2:1
H_BAYER_SAMPLEFACTOR_4 4:1
H_BAYER_SAMPLEFACTOR_8 8:1

4.1.1.10 enum eHBayerPatternOrigin

Pattern origin used for the Bayer conversion.

Enumerator

H_BAYER_PATTERNORIGIN_GB GB-RG.
H_BAYER_PATTERNORIGIN_GR GR-BG.
H_BAYER_PATTERNORIGIN_B BG-GR.
H_BAYER_PATTERNORIGIN_R RG-GB.

4.1.1.11 enum eHInputEvent

Input signal edges.

Enumerator

H_INPUT_EVENT_FALLING_EDGE Input signal edge : falling.
H_INPUT_EVENT_RISING_EDGE Input signal edge : rising.

4.1.1.12 enum eHInputLevel

Input signal levels.

Enumerator

H_INPUT_LEVEL_LOW Input signal level : low.
H_INPUT_LEVEL_HIGH Input signal level : high.
H_INPUT_LEVEL_UNDEFINED Input signal level : unkonwn.

4.1.1.13 enum eHOutputLevel

Output signal levels.

Enumerator

H_OUTPUT_LEVEL_LOW Output signal level : low.
H_OUTPUT_LEVEL_HIGH Output signal level : high.
H_OUTPUT_LEVEL_UNDEFINED Output signal level : undefined.

4.1.1.14 enum eHSpecialCopyFormat

Native support image format by AVI codec.

Enumerator

H_SPECIAL_COPY_JPEG JPEG image data.
H_SPECIAL_COPY_YUV422 YUV422 image.
H_SPECIAL_COPY_UVY422 UVY422 image.
H_SPECIAL_COPY_YUV411 UVY411 image.
H_SPECIAL_COPY_YUV444 UVY444 image.
H_SPECIAL_COPY_RG30 RGB 10 bit image.
H_SPECIAL_COPY_RG48 RGB 12 bit image.
H_SPECIAL_COPY_BYR1 8 bit raw bayer image
H_SPECIAL_COPY_BYR2 10 bit raw bayer image
H_SPECIAL_COPY_BYR4 16 bit raw bayer image
H_SPECIAL_COPY_Y16 16 bit monochrome image
H_SPECIAL_COPY_Y8 8 bit monochrome image
H_SPECIAL_COPY_BGR555 BGR 555 image.
H_SPECIAL_COPY_BGR565 BGR 565 image.
H_SPECIAL_COPY_BGRA BGRx bit image.
H_SPECIAL_COPY_H264 H264 image data.

4.1.1.15 enum eAdvanceSettingsFeatureType

AdvanceSettings support data type.

Enumerator

H_ADVANCE_ENUMERATION Enumeration data type.

H_ADVANCE_INTEGER_VALUE Integer data type.

H_ADVANCE_DOUBLE_VALUE Double data type.

H_ADVANCE_BOOL_VALUE Boolean data type.

H_ADVANCE_COMMAND Command data type.

H_ADVANCE_OTHERS Other data type.

4.1.1.16 enum eFastColorCorrectionMode

Color correction mode.

Enumerator

H_FAST_CORRECTION_NONE No color correction mode.

H_FAST_CORRECTION_MANUAL Manual color correction mode.

H_FAST_CORRECTION_AUTO Auto color correction mode.

4.1.1.17 enum eHImageDateTimeFormat

TIFF files DATETIME formats.

Enumerator

H_DATETIME_STANDARD Standard Time [YYYY:MM:DD HH:MM:SS] : the tiff specification format.

H_DATETIME_FULL Full Time [YYMMDD HHMMSSmmmuuu] : includes milliseconds and microseconds.

H_DATETIME_TIME_T Raw Time [time_t.mmmuuu] : includes the standard UNIX 32-bit time_t value and the milliseconds/microseconds.

H_DATETIME_LTC LTC Time [HH:MM:SS-FF] : The source frames must have been captured with the support of a LTC timing device.

4.1.1.18 enum eHMovieMode

The current mode of a movie file.

AVI & quicktime files that can't do simultaneous recording and playback.

Enumerator

H_MOVIE_UNDEFINED Undefined.

H_MOVIE_RECORDING Recording-only mode.

H_MOVIE_PLAYBACK Playback-only mode.

4.1.1.19 enum eHAviPlaybackFormat

Format of images read from an AVI.

Enumerator

H_AVI_PLAYBACK_NATIVE Playback using the native image format.
H_AVI_PLAYBACK_BGR Convert images to BGR.
H_AVI_PLAYBACK_BGRX Convert images to BGRx.

4.1.1.20 enum eHRotationInterpolationMode

Rotation interpolation mode.

Enumerator

H_ROTATION_INTER_NN Nearest pixel.
H_ROTATION_INTER_LINEAR Bilinear interpolation.
H_ROTATION_INTER_CUBIC Bicubic interpolation.

4.1.1.21 enum eHResizeInterpolationMode

Resize interpolation mode.

Enumerator

H_RESIZE_INTER_NN Nearest neighbor interpolation.
H_RESIZE_INTER_LINEAR Linear interpolation.
H_RESIZE_INTER_CUBIC Cubic interpolation.
H_RESIZE_INTER_LANCZOS Interpolation with Lanczos window.
H_RESIZE_INTER_SUPER Supersampling interpolation.

4.1.1.22 enum eHMotionDetectionMode

Motion detection algorithm type.

Enumerator

H_MOTION_DETECTION_MEAN Mean different.
H_MOTION_DETECTION_ABS_DIFF absolute different
H_MOTION_DETECTION_MAX_ABS_DIFF Max absolute different.
H_MOTION_DETECTION_EDGE Edge detection.
H_MOTION_DETECTION_EDGE_ADAPTIVE Adaptive edge detection.

4.1.1.23 enum eHSobelMode

Sobel edge detection type.

Enumerator

H_SOBEL_VERTICAL Vertical only.
H_SOBEL_HORIZONTAL Horizontal only.
H_SOBEL_BOTH Both side.

4.1.1.24 enum eMonoToPseudoColorLUT

Pseudo color processing algorithm mode.

Enumerator

H_MONO_TO_PSEUDO_COLOR_DISABLED Disbale.

H_MONO_TO_PSEUDO_COLOR_LUT_RAINBOW Raibow.

H_MONO_TO_PSEUDO_COLOR_LUT_INVERTED_RAINBOW Inverse Raibow.

H_MONO_TO_PSEUDO_COLOR_LUT_HOT Hot.

H_MONO_TO_PSEUDO_COLOR_LUT_COLD Cold.

H_MONO_TO_PSEUDO_COLOR_LUT_SATURATION_3 Saturation level 3.

H_MONO_TO_PSEUDO_COLOR_LUT_SATURATION_5 Saturation level 5.

H_MONO_TO_PSEUDO_COLOR_LUT_NEGATIVE Negative.

4.1.1.25 enum eHGRABBER_NOTIFICATION

Notifications from the grabber.

Enumerator

H_GRABBER_NOTIFICATION_CONNECTION_LOST Connection lost.

H_GRABBER_NOTIFICATION_INVALID_SIGNAL Invalid signal.

H_GRABBER_NOTIFICATION_SIGNAL_PRESENT Signal present.

H_GRABBER_NOTIFICATION_CONNECTED Camera connected.

H_GRABBER_NOTIFICATION_DISCONNECTED Camera disconnected.

H_GRABBER_NOTIFICATION_CAPTURE_ERROR Capture error.

H_GRABBER_NOTIFICATION_CAPTURE_EVENT_RECEIVED Event received during capture.

H_GRABBER_NOTIFICATION_TIMESTAP_SET Timestamp set.

H_GRABBER_NOTIFICATION_FRAME_DROP Frame dropped.

H_GRABBER_NOTIFICATION_PTP_CLOCK_STATUS PTP clock status.

4.1.1.26 enum eHSUGGESTED_ACTION

Actions suggested by the grabber along with a notification.

Enumerator

H_SUGGESTED_ACTION_NONE No action.

H_SUGGESTED_ACTION_RELOAD_GRABBER Reload grabber.

H_SUGGESTED_ACTION_UNLOAD_GRABBER Unload grabber.

H_SUGGESTED_ACTION_PAUSE_STREAM Pause streaming.

H_SUGGESTED_ACTION_RESUME_STREAM Resume streaming.

H_SUGGESTED_ACTION_RESTART_STREAM Restart streaming.

H_SUGGESTED_ACTION_OPEN_HARDWARESETTINGDLG Open Hardware Properties dialog.

H_SUGGESTED_ACTION_ENABLE_FRAME_DROP Enable frame drop flag.

H_SUGGESTED_ACTION_DISABLE_FRAME_DROP Disable frame drop flag.

4.1.1.27 enum eHMirror_Mode

Mirroring mode.

Enumerator

H_MIRROR_HORZ Flip left-right.
H_MIRROR_VERT Flip up-down.
H_MIRROR_BOTH Flip left-right & up-down.

4.1.1.28 enum eHMerge_Mode

Image merging mode.

Enumerator

H_MERGE_HORZ Merge along the horizontal axis. (side-by-side)
H_MERGE_VERT Merge along vertical axis. (stacked)

4.1.1.29 enum eAudioFileFormat

Support audio file format.

Enumerator

H_AUDIO_FILE_AUD_PCM AUD audio format.
H_AUDIO_FILE_BWAVE_PCM BWAV audio format.

4.1.1.30 enum eAudFileMode

AUD file data type support.

Enumerator

H_AUD_FILE_AUD_BLOCK Data block mode.
H_AUD_FILE_BWAVE_CONTINOUS Continous data, not support yet.

4.1.1.31 enum eHAudioFormat

Audio format.

Enumerator

H_AUDIO_UNKNOWN Unkown audio format.
H_AUDIO_INT_MSB_PLANAR_PCM MSB integer planar audio format.
H_AUDIO_INT_PLANAR_PCM LSB integer planar audio format.
H_AUDIO_FLOAT_MSB_PLANAR_PCM MSB float planar audio format.
H_AUDIO_FLOAT_PLANAR_PCM LSB float planar audio format.
H_AUDIO_UINT_PLANAR_PCM Unsigned integer packed audio format(8 bit only)
H_AUDIO_INT_PACKED_PCM LSB integer packed audio format.
H_AUDIO_UINT_PACKED_PCM Unsigned integer packed audio format(8 bit only)
H_AUDIO_FLOAT_PACKED_PCM LSB float packed audio format.
H_AUDIO_INT_MSB_PACKED_PCM MSB integer packed audio format.
H_AUDIO_FLOAT_MSB_PACKED_PCM MSB float packed audio format.

4.1.1.32 enum eHAudioCompression

Compressed audio format.

Enumerator

H_AUDIO_COMPRESSION_NONE Uncompressed audio.

H_AUDIO_COMPRESSION_MP3 MP3 audio.

H_AUDIO_COMPRESSION_AAC AAC audio.

4.1.1.33 enum eHAudioPlayEventCode

Event code for audio playback.

Enumerator

H_AUDIO_READ_FAILED Failed to read audio file.

H_AUDIO_UNSUPPORTED_FORMAT Unsupported audio format.

H_AUDIO_SAMPLE_BUFFER_CHANGED Audio sample buffer changed.

4.1.1.34 enum eHExportFormat

Linescan manager exported file format.

Enumerator

H_EXPORT_BMP BMP files.

H_EXPORT_JPG JPG files.

H_EXPORT_TIFF TIFF files.

H_EXPORT_PNG PNG files.

H_EXPORT_FIT FIT files.

H_EXPORT_JP2 JP2 files.

H_EXPORT_DPX DPX files.

H_EXPORT_AVI AVI files.

H_EXPORT_SEQ SEQ files.

H_EXPORT_MOV MOV files.

4.1.1.35 enum eDAQDeviceParamType

DAQ device parameter data type.

Enumerator

H_DAQ_PARAM_ENUM enumeration data type

H_DAQ_PARAM_FLOAT Float data type.

H_DAQ_PARAM_INT Integer data type.

4.1.1.36 enum eTimeSourceParamType

Time source device parameter data type.

Enumerator

H_TS_PARAM_ENUM enumeration data type
H_TS_PARAM_FLOAT Float data type.
H_TS_PARAM_INT Integer data type.

4.1.1.37 enum eHCayer_Compression

Cayer compression type.

Enumerator

H_CYR_VANILLA Lossless.
H_CYR_VANILLA_LOSSY_LOW Lossy with a maximum error of 1 bit per pixel.
H_CYR_VANILLA_LOSSY_MED Lossy with a maximum error of 2 bits per pixel.
H_CYR_VANILLA_LOSSY_HI Lossy with a rare maximum error of 6 bits per pixel.

4.1.1.38 enum eHMetadataFormat

Metadata content format.

Enumerator

H_METADATA_UNKNOWN Unkonwn format.
H_METADATA_BOOL bool (1 byte)
H_METADATA_BYTE byte (1 byte)
H_METADATA_SHORT short (2 bytes)
H_METADATA_USHORT unsigned short (2 bytes)
H_METADATA_INT int (4 bytes)
H_METADATA_UINT unsigned int (4 bytes)
H_METADATA_DOUBLE double (8 bytes)
H_METADATA_STRING wchar_t[] (variable)
H_METADATA_BINARY BYTE[] (variable)
H_METADATA_INT64 __int64 (8 bytes)
H_METADATA_UINT64 unsigned __int64 (8 bytes)

4.1.1.39 enum eDeviceAcclType

Supported GPU type.

Enumerator

H_GPU_DEVICE_UNKNOWN Unkonwn device.
H_CPU_DEVICE_ONLY CPU Processor.
H_GPU_DEVICE_NVIDIA_CUDA NVIDIA CUDA.
H_GPU_DEVICE_NVIDIA_OPENCL NVIDIA OPENCL.
H_GPU_DEVICE_AMD AMD GPU.
H_GPU_DEVICE_INTEL Intel GPU.

4.1.1.40 enum eJpegEncodeAlgoType

JPEG encoding type.

Enumerator

H_JPEG_EN_ALGO_IPP using intel IPP

H_JPEG_EN_ALGO_GPU using GPU

H_JPEG_EN_ALGO_TURBO using Turbo Jpeg lib

4.1.1.41 enum eJpegImageExportType

JPEG exporting type.

Enumerator

H_JPEG_IMAGE_EXPORT_IPP using intel IPP

H_JPEG_IMAGE_EXPORT_GPU using GPU

H_JPEG_IMAGE_EXPORT_TURBO using Turbo Jpeg lib

H_JPEG_IMAGE_EXPORT_FREE_IMAGE using Frame Image lib

4.1.1.42 enum eJpegDecodeAlgoType

JPEG decoding type.

Enumerator

H_JPEG_DE_ALGO_IPP using intel IPP

H_JPEG_DE_ALGO_TURBO using Turbo Jpeg lib

H_JPEG_DE_ALGO_GPU_INTEL using intel GPU

4.1.1.43 enum eHErrorCode

Hermes error codes.

Enumerator

H_ERROR_SUCCESS Success.

H_ERROR_NOAUTHORIZATION Could not find a valid authorization code for this module.

H_ERROR_FAILEDREGISTER Invalid Registration code.

H_ERROR_EXPORT_IN_PROGRESS The export has already begun.

H_ERROR_INVALID_COMPRESSION Invalid JPEG compression method.

H_ERROR_NO_BEGINEXPORT You must call a BeginExport function first.

H_ERROR_NO_IMAGE_DATA No image data.

H_ERROR_INVALID_DIGIT Digit must be between 0 and 12.

H_ERROR_NO_GRABMAN Could not find the grabber manager (Grabman.dll)

H_ERROR_INI_FAIL Grabber Initialization failed.

H_ERROR_CREATE_BUFFER Could not create a buffer.

H_ERROR_BUFFER_INTERNAL CreateBuffer function failed.

H_ERROR_NO_GRABBER No grabber loaded.

H_ERROR_CONVERT_BGR24 Conversion to BGR24 failed.

H_ERROR_CONVERT_BGRX Conversion to BGRx failed.

H_ERROR_CONVERT_UNSUPPORTED Unsupported conversion.

H_ERROR_CONVERT_YUV422 Error conversion Intel YUV422.

H_ERROR_CONVERT_UVY422 Error conversion Intel UVY422.

H_ERROR_CONVERT_UVY411 Error conversion Intel UVY411.

H_ERROR_CONVERT_UVY444 Error conversion Intel UVY444.

H_ERROR_CREATE_SYSTEM_ENUM Failed to create system enumerator.

H_ERROR_CREATE_CLASS_ENUM Failed to create class enumerator.

H_ERROR_MONIKER_NULL Enum moniker null.

H_ERROR_LOADING_VIDEO_CODEC Failed to load the desired video codec.

H_ERROR_LOADING_AUDIO_SOURCE Failed to load the desired audio source.

H_ERROR_LOADING_AUDIO_CODEC Failed to load the desired audio codec.

H_ERROR_NO_CODEC_SETTINGS No settings for this codec.

H_ERROR_CREATE_GRAPH_BUILDER Failed to create the graph builder.

H_ERROR_CREATE_CAPTURE_BUILDER Failed to create the capture graph.

H_ERROR_CREATE_FEEDER Could not create the ImageFeeder Filter.

H_ERROR_ADD_FEEDER Could not add the image feeder to the graph.

H_ERROR_ADD_VCODEC Could not add the video codec to the graph.

H_ERROR_ADD_ACODEC Could not add the audio codec to the graph.

H_ERROR_ADD_AUDIO_SOURCE Could not add the audio source to the graph.

H_ERROR_ADD_MUX_FILE Could not setup MUX and file writer.

H_ERROR_CONNECT_VSOURCE_VCODEC Failed to connect video source to video compressor.

H_ERROR_CONNECT_VCODEC_MUX Failed to connect video compressor to mux.

H_ERROR_CONNECT_VSOURCE_MUX Failed to connect video source to mux.

H_ERROR_CONNECT_ASOURCE_ACODEC Failed to connect audio source to audio compressor.

H_ERROR_CONNECT_ACODEC_MUX Failed to connect audio compressor to mux.

H_ERROR_CONNECT_ASOURCE_MUX Failed to connect audio source to mux.

H_ERROR_CONNECT_SOURCE_SPLITTER Failed to connect the file source to splitter.

H_ERROR_CONNECT_SPLITTER_RENDERER Failed to connect the splitter to the renderer.

H_ERROR_SYNCH_SOURCE Failed to set the synch source.

H_ERROR_INTERLEAVE_MODE Failed to set the interleave mode.

H_ERROR_INTERLEAVE_TIME Failed to set the interleave time.

H_ERROR_SEQ_CANTCREATEFILE Invalid file handle when trying to open for writing.

H_ERROR_NOSEQ No sequence to write to or invalid image.

H_ERROR_WRITE_SEQ Could not write the image to the sequence. (Disk full ?)

H_ERROR_SEQ_INVALIDMAGICNUMBER Invalid Magic Number.

H_ERROR_SEQ_WRONGHEADERSIZE Invalid Sequence Header Size.

H_ERROR_SEQ_INVALIDLENGTH Invalid Sequence Number of Frames Allocated.

H_ERROR_SEQ_INVALIDFRAMERATE Invalid Sequence Frame Rate.

H_ERROR_SEQ_INVALIDDATA Invalid Sequence Data length.

H_ERROR_SEQ_INVALIDWIDTH Invalid Sequence Width.

H_ERROR_SEQ_INVALIDHEIGHT Invalid Sequence Height.

H_ERROR_SEQ_INVALIDDEPTH Invalid Sequence Bit Depth.

H_ERROR_SEQ_INVALIDIMAGESIZE Invalid Sequence Image Size.

H_ERROR_SEQ_UNSUPPORTED_FORMAT This sequence does not support this image format.

H_ERROR_SECTORBYTES_ZERO Bytes per sector is 0.

H_ERROR_FILENOTFOUND File not found.

H_ERROR_CANTCREATEFILE Could not create/open desired file.

H_ERROR_INVALID_INDEX Invalid index.

H_ERROR_OUTOFMEMORY Out of memory.

H_ERROR_IMAGEFORMATCHANGED The image format is incompatible with the current sequence.

H_ERROR_TOOMUCHDATA User data is too large to be written in the sequence.

H_ERROR_GDIDRAW GDI draw failed.

H_ERROR_SCRIPTRUNNING Stop the script before changing it.

H_ERROR_IMAGE_FORMAT Unsupported image format.

H_ERROR_SCRIPTER_NOT_RUNNING The Scripter module is not running.

H_ERROR_STOP_PLAYBACK Must not be in playback mode.

H_ERROR_STREAMING This action can't be done if grabber is streaming.

H_ERROR_DONTSUPPORTREWRITE This sequence format doesn't support rewrite.

H_ERROR_UNSUPPORTED This feature is unsupported.

H_ERROR_UNSUPPORTED_SETTINGS Settings interface not found on driver.

H_ERROR_UNSUPPORTED_ADJUSTMENTS Adjustments interface not found on driver.

H_ERROR_NOCOMPORT No COM PORT found on this grabber.

H_ERROR_EXPORT_FAILED Failed to export image.

H_ERROR_INPUTSUPPORT_LOAD Could not load the input support device.

H_ERROR_OUTPUTSUPPORT_LOAD Could not load the output support device.

H_ERROR_VIDEOOUTPUTSUPPORT_LOAD Could not load the video output support device.

H_ERROR_UNSUPPORTED_ADVANCEFEATURE AdvanceSettings is unsupported.

H_ERROR_NOBITMAP No bitmap file selected.

H_ERROR_QUICKTIME_INIFAIL Quicktime could not be initialized.

H_ERROR_WRONG_MODE This action can't be done while in the current mode.

H_ERROR_QUICKTIME_ERROR A Quicktime function returned an error.

H_ERROR_ROTATION_ERROR This parameter is not supported.

H_ERROR_MIRROR_ERROR Image mirror error.

H_ERROR_RESIZE_ERROR Failed to resize image.

H_ERROR_MERGE_ERROR Image merge error.

H_ERROR_INPUTSUPPORT_DEV_INIT_FAILED Input device initialization failed.

H_ERROR_INPUTSUPPORT_DEV_UNINIT_FAILED Input device uninitialization failed.

H_ERROR_INPUTSUPPORT_MONITOR Failed to start/stop monitoring input line.

H_ERROR_OUTPUTSUPPORT_DEV_INIT_FAILED Output device initialization failed.

H_ERROR_OUTPUTSUPPORT_DEV_UNINIT_FAILED Output device uninitialization failed.

H_ERROR_ADD_FILEREADER Could not add the File Reader to the graph.

H_ERROR_ADD_AVISPLITTER Could not add the AVI Splitter to the graph.

H_ERROR_CREATE_RENDER Could not create Image Renderer Filter.

H_ERROR_NO_PROCESSING_APPLIED No processings were applied on the image.

H_ERROR_COLOR_BALANCING The image could not be color balanced.

H_ERROR_BAYER_CONVERSION The image could not be Bayer converted.

H_ERROR_DAQSUPPORT_LOAD Could not load DAQ module.

H_ERROR_DAQSUPPORT_READ_FAILED Could not read daq data.

H_ERROR_DAQSUPPORT_CANTCREATEFILE Invalid DAQ file handle when trying to open for writing.

H_ERROR_DAQSUPPORT_INVALID_INDEX Invalid DAQ index.

H_ERROR_DAQSUPPORT_NODAQ No DAQ data to write to or invalid DAQ data.

H_ERROR_DAQSUPPORT_WRITE_DAQ Could not write the daq to the file (Disk full ?)

H_ERROR_DAQSUPPORT_FORMAT_CHANGED Daq format changed.

H_ERROR_AUDIO_FORMAT Unsupported audio format.

H_ERROR_AUDIO_LOAD Failed to load audio drvier.

H_ERROR_AUDIO_INPUTSUPPORT_LOAD Failed to load audio input device.

H_ERROR_AUDIO_OUTPUTSUPPORT_LOAD Failed to load audio output device.

H_ERROR_AUDIO_INPUTSUPPORT Doesn't support audio input device.

H_ERROR_AUDIO_OUTPUTSUPPORT Doesn't support audio output device.

H_ERROR_AUDIO_CANTCREATEFILE Invalid Audio file handle when trying to open for writing.

H_ERROR_AUDIO_NOAUD No audio data to write to or invalid audio.

H_ERROR_AUDIO_WRITE_AUD Could not write the audio to the DAQ file (Disk full ?)

H_ERROR_AUDIO_FORMAT_CHANGED Audio format changed.

H_ERROR_AUDIO_INVALID_INDEX Invalid AUD index.

H_ERROR_AUDIO_INVALID_FORMAT Invalid Audio format.

H_ERROR_AUDIO_CHANNEL_MASK_MATCH Audio channel npot match with audio channel number.

H_ERROR_AUDIO_READ_FAILED Audio data reading failed.

H_ERROR_CANT_ADD_FRAME Cannot add more frames to the AVI.

H_ERROR_UNSUPPORT_FUNCTION Don't support this function.

H_ERROR_CREATE_SOCKET Failed to create socket.

H_ERROR_BIND_SOCKET Failed to bind socket.

H_ERROR_LISTEN_SOCKET Failed to listen socket.

H_ERROR_SHUTDOWN_SOCKET Failed to disable sends/receives on socket.

H_ERROR_CLOSE_SOCKET Failed to close socket.

H_ERROR_INVALID_SOCKET Invalid socket.

H_ERROR_SERVER_NOT_RUNNING Server not running.

H_ERROR_SERVER_ALREADY_OPEN Server already open.

H_ERROR_CLIENT_ALREADY_CONNECTED Client already connected.

H_ERROR_CONNECTION_FAILED Connection failed.

H_ERROR_SEND Failed to send data.

H_ERROR_OUT_OF_RANGE Out of range.

H_ERROR_NO_FILE No file.

H_ERROR_H264_UNSUPPOT_IMAGE_SIZE Unsupported image size for H.264 encoding. Width should be factor of 16 and height factor of 8.

H_ERROR_H264_ENCODING H.264 encoding error.

H_ERROR_H264_DECODING H.264 decoding error.

H_ERROR_CREATE_SAMPLE_GRABBER Failed to create sample grabber.

H_ERROR_SAMPLE_GRABBER_CALLBACK Failed to set sample grabber callback.

H_ERROR_ADD_SAMPLE_GRABBER Failed to add sample grabber to the graph.

H_ERROR_CONNECT_ASOURCE_ALEVEL Failed to connect audio source to audio level filter.

H_ERROR_CONNECT_ALEVEL_ACODEC Failed to connect audio level to audio codec.

H_ERROR_CONNECT_ALEVEL_MUX Failed to connect audio level to muxer.

H_ERROR_NO_AUDIO_SOURCE No audio source present.

H_ERROR_NO_RECORDING Currently not in recording mode.

H_ERROR_NO_CAPTURE_GRAPH No capture graph created.

H_ERROR_FAILED_TO_ADD_TAG Failed to add tag.

H_ERROR_FRAME_SKIPPED Key frame not found, frame skipped.

H_ERROR_AVI_TAG_NOT_FOUND AVI tag not found.

H_ERROR_REGISTRY_READ An error occured trying to reading from the registry.

H_ERROR_REGISTRY_WRITE An error occured trying to writing to the registry.

H_ERROR_CREATEC_UNSUPPORT_FORMAT Unsupported format. Createc compression only supports 8 bit mono or mono bayer format.

H_ERROR_INVALID_LUT_FILE Invalid LUT file.

H_ERROR_NOT_CONNECTED Not connected.

H_ERROR_SOCKET_ALREADY_CREATED Socket already created.

H_ERROR_WRITE_ERROR Write error.

H_ERROR_CREATE_FILE_ERROR Failed to create file.

H_ERROR_MALFORMED_XML_FILE Malformed XML file.

H_ERROR_MP4_VIDEO_READ Fail to Read MP4 video data.

H_ERROR_MP4_AUDIO_READ Fail to Read MP4 audio data.

H_ERROR_MPEG4_ENCODING MPEG4 encoding error.

H_ERROR_MPEG4_DECODING MPEG4 decoding error.

H_ERROR_HEVC_ENCODING HEVC encoding error.

H_ERROR_HEVC_DECODING HEVC decoding error.

H_ERROR_DECODE_MORE_DATA Decoder need more frame.

H_ERROR_AUIDO_PROCESS_MORE_DATA Audio processing need more data.

H_ERROR_AUIDO_PROCESS_NO_DATA Audio processing no more data.

H_ERROR_AAC_DECODING AAC decoding error.

H_ERROR_MP3_DECODING MP3 decoding error.

H_ERROR_UNSUPPORTED_INTERFACE Unsupported interface.

H_ERROR_GPU_IMAGE_PUSH Fail to push image to GPU.

4.2 HermesUtils.h File Reference

```
#include "HEnums.h"
#include <windows.h>
#include <TCHAR.h>
#include "importexport.h"
```

Functions

- HERMES_API LPCTSTR HImageFormatString (eHImageFormat format)

  Retreive the string describing an image format ID.
- HERMES_API LPCTSTR HImageCompressionString (eHCompression compression)

  Retreive the string describing a compression format ID.
- HERMES_API bool IsFormatMono (eHImageFormat format)

  Check if a given image format is a kind of monochrome image.
- HERMES_API bool IsFormatToF (eHImageFormat format)

  Check if a given image format is a kind of ToF image.

- HERMES_API void AbsoluteToRelative (long reference, unsigned short referenceMS, unsigned short referenceUS, long absolute, unsigned short absoluteMS, unsigned short absoluteUS, long &relative, unsigned short &relativeMS, unsigned short &relativeUS)

  Convert an absolute time to a relative time given a reference time.
- HERMES_API void RelativeToAbsolute (long reference, unsigned short referenceMS, unsigned short referenceUS, long relative, unsigned short relativeMS, unsigned short relativeUS, long &absolute, unsigned short &absoluteMS, unsigned short &absoluteUS)

  Convert a relative time to an absolute time a reference time.
- HERMES_API bool ComputeTimeDifference (long time1, unsigned short time1MS, unsigned short time1U↩S, long time2, unsigned short time2MS, unsigned short time2US, long &diffTime, short &diffTimeMS, short &diffTimeUS, bool &time2IsGreater)

  Compute the time delta between two absolute times.
- HERMES_API int CompareTimestamps (long time1, unsigned short time1MS, unsigned short time1US, long time2, unsigned short time2MS, unsigned short time2US)

  Compare two times to check which one is the greater.
- HERMES_API void AddTimestampOffset (long &time, short &timeMS, short &timeUS, long offset, short offsetMS, short offsetUS)

  Add a time offset to an existing time value.
- HERMES_API void CreateDirectoryRecursive (LPCTSTR filepath)

  Make sure every folders in the given path exist and create them otherwise.
- HERMES_API double ReadAVIFrameRate (LPCTSTR filePath)

  Read the frame rate value from an AVI file.
- HERMES_API void WriteAVIFrameRate (LPCTSTR filePath, double framerate)

  Overwrite the frame rate of an AVI file.

4.3 MetadataID.h File Reference

This file hold the unique identifiers for all generic metadata types along with all the metadata types created by NorPix.

Macros

ALLOWED RANGE

- #define **UID_METADATA_GENERIC_FIRST** 0x00000000
- #define **UID_METADATA_GENERIC_LAST** 0x00000020
- #define **UID_METADATA_NORPIX_FIRST** 0x00000021
- #define **UID_METADATA_NORPIX_LAST** 0x7FFFFFFF
- #define **UID_METADATA_USER_FIRST** 0x80000000
- #define **UID_METADATA_USER_LAST** 0xFFFFFFFF

GENERIC TYPES

- #define **UID_METADATA_GENERIC_BOOL** (UID_METADATA_GENERIC_FIRST+0)
- #define **UID_METADATA_GENERIC_BYTE** (UID_METADATA_GENERIC_FIRST+1)
- #define **UID_METADATA_GENERIC_SHORT** (UID_METADATA_GENERIC_FIRST+2)
- #define **UID_METADATA_GENERIC_USHORT** (UID_METADATA_GENERIC_FIRST+3)
- #define **UID_METADATA_GENERIC_INT** (UID_METADATA_GENERIC_FIRST+4)
- #define **UID_METADATA_GENERIC_UINT** (UID_METADATA_GENERIC_FIRST+5)
- #define **UID_METADATA_GENERIC_DOUBLE** (UID_METADATA_GENERIC_FIRST+6)
- #define **UID_METADATA_GENERIC_STRING** (UID_METADATA_GENERIC_FIRST+7)
- #define **UID_METADATA_GENERIC_BINARY** (UID_METADATA_GENERIC_FIRST+8)
- #define **UID_METADATA_GENERIC_INT64** (UID_METADATA_GENERIC_FIRST+9)
- #define **UID_METADATA_GENERIC_UINT64** (UID_METADATA_GENERIC_FIRST+10)

NORPIX DEFINED TYPES

- #define **UID_METADATA_NORPIX_TIME64** (UID_METADATA_NORPIX_FIRST+0)
- #define **UID_METADATA_NORPIX_LTC** (UID_METADATA_NORPIX_FIRST+1)
- #define **UID_METADATA_NORPIX_GPS** (UID_METADATA_NORPIX_FIRST+2)
- #define **UID_METADATA_NORPIX_GPS_EX** (UID_METADATA_NORPIX_FIRST+3)
- #define **UID_METADATA_NORPIX_LIDAR** (UID_METADATA_NORPIX_FIRST+4)
- #define **UID_METADATA_NORPIX_KINECT_SKELETON_FRAME** (UID_METADATA_NORPIX_FIR↩ ST+5)
- #define **UID_METADATA_NORPIX_GRT_HEADER** (UID_METADATA_NORPIX_FIRST+6)
- #define **UID_METADATA_NORPIX_DMI** (UID_METADATA_NORPIX_FIRST+7)
- #define **UID_METADATA_NORPIX_DMI_FB** (UID_METADATA_NORPIX_FIRST+8)
- #define **UID_METADATA_NORPIX_INPUT_LINE_STATE** (UID_METADATA_NORPIX_FIRST+9)
- #define **UID_METADATA_NORPIX_CAMERA_INFO** (UID_METADATA_NORPIX_FIRST+10)
- #define **UID_METADATA_NORPIX_IMAGE_STATISTICS** (UID_METADATA_NORPIX_FIRST+11)

4.3.1 Detailed Description

This file hold the unique identifiers for all generic metadata types along with all the metadata types created by NorPix.

A third party developper can create new metadata types using one of the ID in the range of [UID_METADATA_U↩ SER_FIRST, UID_METADATA_USER_LAST]

Unique identifiers 0 to 2147483647 are reserved for generic and NorPix metadatas. Third party users can create their own metadatas using IDs from 2147483648 to 4294967295.

Index

AVI File, 13
AbsDiffDetect
    HMotionDetection, 315, 321
AbsMaxDiffDetect
    HMotionDetection, 317, 322
AbsoluteToRelative
    Utilities, 19
Accessible
    HGrabModule, 176
Acquisition, 7
AdaptiveEdgeDetect
    HMotionDetection, 318, 323
AddAudio
    HMP4File, 332
    HMPEGTsFile, 340
AddData
    HAACFile, 39
    HMP3File, 326
AddFrame
    HAviFile, 97
    HMP4File, 332
    HMPEGTsFile, 339
AddJPEGMetadata
    HImageExporter, 222
AddMetadata
    HImage, 193
AddThirdPartLutFile
    HImageColorProcessing, 206
AddTimestampOffset
    Utilities, 21
AddVirtualURL
    HStreamDeliver, 375
AllocateAverageBuffer
    HFrameAverage, 143, 144
ApplyColorBalance
    HWhiteBalance, 403
ApplyColorProcessing
    HImageColorProcessing, 197
AreChannelsOpen
    HAudioRecorder, 84
Audio, 9
AudioInputPresent
    HAudioDriverASIO, 48
    HAudioDriverDirectX, 58
AudioOutputPresent
    HAudioDriverASIO, 52
    HAudioDriverDirectX, 62

BeginDrawing
    HImageOverlay, 233
BeginExport
    HAviExporter, 90
BeginExportBmp
    HImageExporter, 213
BeginExportBmpTo
    HImageExporter, 217
BeginExportDng
    HImageExporter, 216
BeginExportDngTo
    HImageExporter, 219
BeginExportDpx
    HImageExporter, 216
BeginExportDpxTo
    HImageExporter, 219
BeginExportFit
    HImageExporter, 215
BeginExportFitTo
    HImageExporter, 218
BeginExportJpeg
    HImageExporter, 214
BeginExportJpeg2k
    HImageExporter, 215
BeginExportJpeg2kTo
    HImageExporter, 218
BeginExportJpegTo
    HImageExporter, 218
BeginExportJpegXR
    HImageExporter, 216
BeginExportJpegXRTo
    HImageExporter, 219
BeginExportPng
    HImageExporter, 215
BeginExportPngTo
    HImageExporter, 218
BeginExportTiff
    HImageExporter, 214
BeginExportTiffTo
    HImageExporter, 218
Broadcast
    HTcpServer, 387
BuildFrom
    HMetadata, 271
BuildMetadataString
    HMetadataGenericBool, 275
    HMetadataGenericByte, 276
    HMetadataGenericDouble, 277
    HMetadataGenericInt, 279
    HMetadataGenericInt64, 280
    HMetadataGenericShort, 281

HMetadataGenericString, 283
HMetadataGenericUInt, 284
HMetadataGenericUInt64, 285
HMetadataGenericUShort, 287
HMetadataNorPixCameraInfo, 292
HMetadataNorPixDMIExtended, 295
HMetadataNorPixDMIFBExtended, 296
HMetadataNorPixGPS, 297
HMetadataNorPixGPSExtended, 301
HMetadataNorPixGRTHeader, 303
HMetadataNorPixImageStatistics, 305
HMetadataNorPixInputLineState, 306
HMetadataNorPixLTC, 309
HMetadataNorPixLidar, 308
HMetadataNorPixTime64, 310

CAudioCaptureCallback, 31
OnDataReceived, 31
CAudioPlaybackCallback, 31
OnAudioSourceEvent, 32
OnReadNewData, 32
CAudioPlayerCallback, 32
OnEndPlayback, 33
OnPlayBuffer, 33
CAudioRecorderCallback, 33
OnBufferReceived, 33
OnBufferSaved, 33
CCineFormHDParam, 34
CCineFormParam, 34
CCineFormRawParam, 35
CDaqSupportCallback, 35
OnDaqDeviceStatusChanged, 36, 37
OnDataReceived, 36
WantContinuousDataRead, 35, 36
CGPSDataStruct, 37
CheckGrayConvertionImageFormat
HGrayConverter, 185, 186
CheckImageMergeFormat
HImageMerge, 224
CheckImageMirrorFormat
HImageMirror, 227
CheckImageReSampleFormat
HImageResize, 242
CheckImageResizeFormat
HImageResize, 241
CloneAudioData
HAudioData, 43
CloneDaqDataSet
HDaqDataSet, 114, 117
CloneHImage
HImage, 189
Close
HAACFile, 41
HAudioFile, 70
HDaqFile, 123, 130
HMP3File, 328
HMP4File, 333
HMPEGTsFile, 340
HSequence, 354
CloseAdvanceSettingsWindow
HGrabModule, 175
CloseAudioInputDevice
HAudioDriverASIO, 49
HAudioDriverDirectX, 59
CloseAudioOutputDevice
HAudioDriverASIO, 53
HAudioDriverDirectX, 63
CloseAviFile
HAviFile, 97
CloseChannels
HAudioRecorder, 83
CloseConnection
HTcpServer, 385
CloseServer
HTcpServer, 385
CompareTimestamps
Utilities, 21
ComputeColorBalance
HWhiteBalance, 401, 403
ComputeTimeDifference
Utilities, 20
Configure
HLineScanManager, 262, 265
ConfigureAVIExport
HLineScanManager, 264, 267
ConfigureImageExport
HLineScanManager, 264, 267
ConfigureMOVExport
HLineScanManager, 264, 267
Connect
HUdpClient, 391
ConnectTo
HTcpClient, 381
ConvertToBGR
HImageConverter, 210
ConvertToBGRx
HImageConverter, 211
ConvertToGray
HMotionDetection, 315, 321
ConvertToMono8
HImageConverter, 211
Create
HAACFile, 39
HAudioFile, 69
HDaqFile, 123, 129
HMP3File, 325
HMP4File, 330
HMPEGTsFile, 338
HSequence, 353
CreateAdvanceSettingsWindow
HGrabModule, 174
CreateAudioData
HAudioData, 43
CreateAviFile
HAviFile, 95, 96
CreateDaqDataSet
HDaqDataSet, 114, 117

CreateDirectoryRecursive
Utilities, 21
CreateHImage
HImage, 188

DaqPresent
HIODriver, 251
Data (DAQ), 10
Decode
HJpegCompression, 260
DeleteDaqDataSet
HDaqDataSet, 115, 117
DeleteHImage
HImage, 189
DeliverNewAudio
HStreamDeliver, 375
DeliverNewFrame
HStreamDeliver, 375
DetectVideoCodecType
HAviFile, 100
DisableAMDOpenCLEncoding
HGPUDeviceManager, 149
DisableNvidiaCudaEncoding
HGPUDeviceManager, 148
DisableNvidiaOpenCLEncoding
HGPUDeviceManager, 147
DisableVideoOutput
HVideoOutputDriver, 396
Disconnect
HTcpClient, 381
DoModalAdvanceSettingsWindow
HGrabModule, 174
DumpHeader
HSequence, 356

eAdvanceSettingsFeatureType
HEnums.h, 420
eAudFileMode
HEnums.h, 424
eAudioFileFormat
HEnums.h, 424
eDAQDeviceParamType
HEnums.h, 425
eDeviceAcclType
HEnums.h, 426
eFastColorCorrectionMode
HEnums.h, 421
eHAudioCompression
HEnums.h, 424
eHAudioFormat
HEnums.h, 424
eHAudioPlayEventCode
HEnums.h, 425
eHAviPlaybackFormat
HEnums.h, 421
eHBayerPatternOrigin
HEnums.h, 419
eHBayerSampleFactor
HEnums.h, 419
eHCayer_Compression
HEnums.h, 426
eHCompression
HEnums.h, 418
eHDescriptionFormat
HEnums.h, 418
eHErrorCode
HEnums.h, 427
eHExportFormat
HEnums.h, 425
eHGRABBER_NOTIFICATION
HEnums.h, 423
eHGrabberInfoType
HEnums.h, 416
eHImageDateTimeFormat
HEnums.h, 421
eHImageFormat
HEnums.h, 417
eHInputEvent
HEnums.h, 419
eHInputLevel
HEnums.h, 420
eHJpegQuality
HEnums.h, 417
eHMerge_Mode
HEnums.h, 424
eHMetadataFormat
HEnums.h, 426
eHMirror_Mode
HEnums.h, 423
eHMotionDetectionMode
HEnums.h, 422
eHMovieMode
HEnums.h, 421
eHOutputLevel
HEnums.h, 420
eHProtectionStatus
HEnums.h, 419
eHResizeInterpolationMode
HEnums.h, 422
eHRotationInterpolationMode
HEnums.h, 422
eHSUGGESTED_ACTION
HEnums.h, 423
eHSobelMode
HEnums.h, 422
eHSpecialCopyFormat
HEnums.h, 420
eHSupportedModule
HEnums.h, 413
eJpegDecodeAlgoType
HEnums.h, 427
eJpegEncodeAlgoType
HEnums.h, 426
eJpegImageExportType
HEnums.h, 427
eMonoToPseudoColorLUT
HEnums.h, 422

eTimeSourceParamType
HEnums.h, 425
EdgeDetect
HMotionDetection, 317, 322
Enable3DLookupTable
HImageColorProcessing, 204
EnableAMDOpenCLEncoding
HGPUDeviceManager, 148
EnableAutoColorBalance
HImageColorProcessing, 202
EnableBGRxConversion
HImageColorProcessing, 199
EnableBayerConversion
HImageColorProcessing, 197
EnableColorProcessingHQConversion
HImageColorProcessing, 200
EnableGPUJpegCompressionBayerInterp
HImageExporter, 221
HSequence, 370
EnableGPUJpegCompressionColorprcessingLUT
HImageExporter, 222
HSequence, 371
EnableLUTProcess
HImageColorProcessing, 204
EnableMatrixColorCorrection
HImageColorProcessing, 200
EnableMetadata
HSequence, 359
EnableMonochromeConversion
HImageColorProcessing, 209
EnableNvidiaCudaEncoding
HGPUDeviceManager, 148
EnableNvidiaOpenCLEncoding
HGPUDeviceManager, 147
EnablePassthrough
HAudioRecorder, 85
EnableTIFFMetadata
HImageExporter, 222
EnableVideoOutput
HVideoOutputDriver, 396
Encode
HJpegCompression, 259
EndBench
HFPSMonitor, 140
EndDrawing
HImageOverlay, 233
EndExport
HAviExporter, 90
HImageExporter, 217
EndExportTo
HImageExporter, 220
EndQueryMode
HGrabModule, 181
EndRecording
HAACFile, 39
HMP3File, 326
HMP4File, 333
HMPEGTsFile, 340
ExecuteCommandFeature
HGrabModule, 180
Export
HLineScanManager, 265, 268
ExportGrabberSettings
HGrabModule, 160
ExportImage
HAviExporter, 90
HImageExporter, 217
ExportImageTo
HImageExporter, 220
ExportLUTToTextFile
HImageColorProcessing, 208

File Formats, 12
FinalizeWrite
HMP4File, 333
HMPEGTsFile, 340
ForceFrameRate
HAviExporter, 90
HAviFile, 101
ForceOutputBRG24
HImageMerge, 224
FreeAverageBuffer
HFrameAverage, 143, 144

Generic Metadata, 24
Get1DLut
HImageColorProcessing, 209
Get1DPreDefinedLUT
HImageColorProcessing, 207
Get1DPreDefinedLUTCount
HImageColorProcessing, 207
Get1DSupportThirdPartLutFileCount
HImageColorProcessing, 205
Get1DSupportThirdPartLutFileFormatInfo
HImageColorProcessing, 205
Get3DSupportThirdPartLutFileCount
HImageColorProcessing, 205
Get3DSupportThirdPartLutFileFormatInfo
HImageColorProcessing, 205
GetAMDGPUDeviceInfo
HGPUDeviceManager, 148
GetASCSettingsParam
HImageColorProcessing, 203
GetAbsoluteThresholdValue
HMotionDetection, 313, 320
GetActionMode
HMP4File, 335
GetAddress
HTcpServer, 386
GetAdjustmentCaps
HGrabModule, 167
GetAdjustmentCount
HGrabModule, 167
GetAlgorithm
HWhiteBalance, 400
GetAllocatedAudioDatas
HAudioFile, 71

GetAllocatedDataSets
HDaqFile, 125, 131
GetAllocatedFrames
HMP4File, 336
HSequence, 358
GetAlpha
HImageOverlay, 230
GetAperture
HMetadataNorPixCameraInfo, 292
GetAudioBitDepth
HAudioData, 45
GetAudioBitDepthReal
HAudioData, 45
GetAudioChannelCount
HAudioFile, 68
GetAudioCodecCount
HAviFile, 99
GetAudioCodecList
HAviExporter, 89
GetAudioCodecName
HAviFile, 99
GetAudioDataBuffer
HAudioData, 44
GetAudioDepth
HAudioFile, 68
GetAudioDepthReal
HAudioFile, 68
GetAudioFormat
HAudioFile, 68
GetAudioInfo
HAACFile, 40
HMP3File, 327
HMP4File, 334
GetAudioInputChannelCount
HAudioDriverASIO, 50
GetAudioInputChannelInfoByChannelIndex
HAudioDriverASIO, 51
GetAudioInputDeviceCount
HAudioDriverASIO, 48
HAudioDriverDirectX, 58
GetAudioInputDeviceName
HAudioDriverASIO, 48
HAudioDriverDirectX, 59
GetAudioInputFormat
HAudioDriverDirectX, 61
GetAudioInputFormatCount
HAudioDriverDirectX, 60
GetAudioInputFormatInfo
HAudioDriverDirectX, 60
GetAudioInputLevel
HAudioRecorder, 84
GetAudioInputSampleRate
HAudioDriverASIO, 50
GetAudioInputSampleRateInfo
HAudioDriverASIO, 50
GetAudioInputSampleRateInfoCount
HAudioDriverASIO, 49
GetAudioOutputChannelCount
HAudioDriverASIO, 54
GetAudioOutputChannelInfoByChannelIndex
HAudioDriverASIO, 55
GetAudioOutputDeviceCount
HAudioDriverASIO, 52
HAudioDriverDirectX, 62
GetAudioOutputDeviceName
HAudioDriverASIO, 52
HAudioDriverDirectX, 62
GetAudioOutputFormat
HAudioDriverDirectX, 64
GetAudioOutputFormatCount
HAudioDriverDirectX, 63
GetAudioOutputFormatInfo
HAudioDriverDirectX, 64
GetAudioOutputSampleBufferSize
HAudioDriverDirectX, 63
GetAudioOutputSamplePerBuffer
HAudioDriverASIO, 53
GetAudioOutputSampleRate
HAudioDriverASIO, 54
GetAudioOutputSampleRateInfo
HAudioDriverASIO, 54
GetAudioOutputSampleRateInfoCount
HAudioDriverASIO, 53
GetAudioSamplesPerBuffer
HAudioFile, 68
GetAudioSourceCount
HAviFile, 98
GetAudioSourceList
HAviExporter, 88
GetAudioSourceName
HAviFile, 99
GetAufioFileType
HAudioFile, 69
GetAuthor
HAviFile, 106
GetAutoColorBalanceAlgorithm
HImageColorProcessing, 202
GetAutoColorBalanceMode
HWhiteBalance, 401
GetAutoRecoveryIndexStream
HSequence, 367
GetAutoWhiteBalanceROI
HImageColorProcessing, 203
GetAutomaticAdjustment
HGrabModule, 169
GetAviMode
HAviFile, 97
GetBayerPattern
HSequence, 369
GetBenchTime
HFPSMonitor, 140
GetBinning
HGrabModule, 165
GetBitmapFile
HBitmapOverlay, 109
GetBlueChannel

HImage, 191
GetBoolFeatureValue
HGrabModule, 179
GetBufferCount
HAudioRecorder, 85
HGrabModule, 182
GetBufferUsage
HAudioRecorder, 85
HGrabModule, 182
GetCalcMode
HImageColorProcessing, 199
GetCaptureBufferUsage
HAudioDriverASIO, 52
HAudioDriverDirectX, 62
GetCategoriesCap
HGrabModule, 175
GetCategoriesCount
HGrabModule, 175
GetCayerCompressionType
HSequence, 366
GetCenterCoordinate
HRotation, 343, 346
GetChannelNum
HAudioData, 44
HDaqDataSet, 115, 118
GetColorBalance
HWhiteBalance, 400
GetColorBalanceApplied
HWhiteBalance, 401
GetColorClipLevel
HImageColorProcessing, 207
GetCommunicationType
HGrabModule, 173
GetCompressSeqIndexType
HSequence, 366
GetCompressionBayerPattern
HSequence, 366
GetConnectionCount
HTcpServer, 385
GetCopyright
HAviFile, 105
GetCorrectionMatrix
HImageColorProcessing, 201
GetCreationDateTime
HAviFile, 105
GetCrosshairThickness
HImageOverlay, 232
GetCurrentExportedFrame
HLineScanManager, 265, 268
GetCurrentGrabber
HGrabModule, 160
GetCurrentLevel
HIODriver, 247
GetCurrentPosition
HAviFile, 103
GetCurrentValues
HGrabModule, 166
GetDaqDeviceCount
HIODriver, 251
GetDaqDeviceEnumParameterItemCount
HIODriver, 255
GetDaqDeviceEnumParameterItemInfo
HIODriver, 255
GetDaqDeviceEnumParameterValue
HIODriver, 256
GetDaqDeviceFloatParameterItemInfo
HIODriver, 255
GetDaqDeviceFloatParameterValue
HIODriver, 256
GetDaqDeviceIntParameterItemInfo
HIODriver, 256
GetDaqDeviceIntParameterValue
HIODriver, 257
GetDaqDeviceLineCount
HIODriver, 252
GetDaqDeviceLineInfo
HIODriver, 252
GetDaqDeviceLineRangeSettingsCount
HIODriver, 252
GetDaqDeviceLineRangeSettingsInfo
HIODriver, 253
GetDaqDeviceName
HIODriver, 251
GetDaqDeviceParametersCount
HIODriver, 254
GetDaqDeviceParametersInfo
HIODriver, 254
GetData
HMetadata, 272
GetDataSizeBytes
HMetadata, 271
GetDateTimeFormat
HImageExporter, 222
GetDateTimeOriginal
HAviFile, 106
GetDescription
HItem, 258
HSequence, 367
GetDescriptionFormat
HSequence, 368
GetDestinationImageSize
HRotation, 343, 346
GetDestinationSize
HImageResize, 241
GetDeviceInputLineCount
HIODriver, 246
GetDeviceInputLineInfo
HIODriver, 247
GetDeviceName
HMetadataNorPixInputLineState, 307
GetDeviceOutputLineCount
HIODriver, 249
GetDeviceOutputLineInfo
HIODriver, 249
GetDoubleAdjustment
HGrabModule, 168

GetDoubleFeatureCap
HGrabModule, 179
GetDoubleFeatureValue
HGrabModule, 179
GetDriverCount
HGrabModule, 158
GetDriverInfo
HGrabModule, 159
GetDriverName
HAudioDriverASIO, 48
HAudioDriverDirectX, 58
HIODriver, 245
GetEndChar
HGrabModule, 172
GetEnumFeatureValue
HGrabModule, 177
GetEnumFeatureValueCap
HGrabModule, 177
GetEnumFeatureValueCount
HGrabModule, 177
GetErrorString
HErrorManager, 138
GetExposure
HMetadataNorPixCameraInfo, 292
GetFeatureCap
HGrabModule, 176
GetFeatureCount
HGrabModule, 176
GetFirstSocket
HTcpServer, 387
GetFocalLength
HMetadataNorPixCameraInfo, 292
GetFormat
HAudioData, 44
GetFrameCount
HAviFile, 103
GetFrameRate
HSequence, 356
GetGPSData
HGrabModule, 171
GetGain
HMetadataNorPixCameraInfo, 293
GetGamma
HMetadataNorPixCameraInfo, 293
GetGrabberCount
HGrabModule, 181
GetGrabberInfoCount
HGrabModule, 162
GetGrabberInfoName
HGrabModule, 162
GetGrabberInfoValueDouble
HGrabModule, 163
GetGrabberInfoValueLong
HGrabModule, 163
GetGrabberInfoValueString
HGrabModule, 163
GetGrabberInfoValueType
HGrabModule, 163
GetGrabberName
HGrabModule, 181
GetGreenChannel
HImage, 190
GetID
HItem, 258
GetIODriver
HGrabModule, 180
GetIPAddress
HUdpServer, 394
GetImageBitDepth
HGrabModule, 170
HImage, 189
HSequence, 364
GetImageBitDepthReal
HGrabModule, 170
HImage, 190
HSequence, 364
GetImageFormat
HGrabModule, 170
HImage, 190
HSequence, 365
GetImageFromSeq
HLineScanManager, 263, 266
GetImageHeight
HGrabModule, 170
HImage, 189
HSequence, 364
GetImageSizeBytes
HGrabModule, 170
HImage, 190
HSequence, 364
GetImageWidth
HGrabModule, 170
HImage, 189
HSequence, 364
GetIndexAt
HAviFile, 103
HMP4File, 335
HSequence, 358
GetIndexNear
HSequence, 358
GetInputDeviceCount
HIODriver, 245
GetInputDeviceName
HIODriver, 246
GetInputLevel
HMetadataNorPixInputLineState, 307
GetIntegerAdjustment
HGrabModule, 168
GetIntegerFeatureCap
HGrabModule, 178
GetIntegerFeatureValue
HGrabModule, 178
GetIntelGPUDeviceInfo
HGPUDeviceManager, 147
GetInterpolateMode
HRotation, 345, 347

GetInterpolationMode
HImageResize, 241
GetJPEGQuality
HImageExporter, 221
GetJpegDecodeType
HSequence, 370
GetJpegEncodeType
HSequence, 370
GetKeyFrameByIndex
HMP4File, 336
GetLTC
HMetadataNorPixLTC, 309
GetLastError
HAACFile, 39
HAudioDriverASIO, 48
HAudioDriverDirectX, 58
HAudioFile, 68
HAudioPlayer, 74
HAudioRecorder, 80
HAviExporter, 87
HAviFile, 94
HBitmapOverlay, 109
HDaqFile, 123, 129
HFPSMonitor, 139
HFrameAverage, 143, 144
HGPUDeviceManager, 147
HGrabModule, 158
HGrayConverter, 185, 186
HImageColorProcessing, 196
HImageConverter, 210
HImageExporter, 213
HImageMerge, 224
HImageMirror, 226
HImageOverlay, 229
HImageResize, 240
HJpegCompression, 259
HMP3File, 325
HMP4File, 330
HMPEGTsFile, 338
HMotionDetection, 312, 319
HRotation, 342, 346
HSequence, 353
HTcpClient, 380
HTcpServer, 384
HUdpClient, 391
HUdpServer, 393
HViewer, 397
HWhiteBalance, 400
GetLineColor
HImageOverlay, 233
GetLineName
HMetadataNorPixInputLineState, 307
GetLineThickness
HImageOverlay, 232
GetLiveImage
HLineScanManager, 263, 266
GetManualColorBalance
HImageColorProcessing, 202
GetMaxBufferUsage
HAudioRecorder, 85
HGrabModule, 182
GetMaxPlaybackInterval
HSequence, 362
GetMaxResolution
HGrabModule, 165
GetMaxSampleRate
HIODriver, 253
GetMergeMode
HImageMerge, 224
GetMetadata
HDaqFile, 129, 136
HImage, 192
GetMetadataCount
HImage, 192
GetMetadataLength
HDaqFile, 129, 135
GetMetadataString
HMetadata, 272
GetMetadataTypeByID
HMetadataManager, 290
GetMetadataTypeByIndex
HMetadataManager, 290
GetMetadataTypeCount
HMetadataManager, 290
GetMirrorMode
HImageMirror, 226
GetMonoPseudoColorMode
HImageColorProcessing, 207
GetMotionDetectionMode
HMotionDetection, 312, 319
GetNextPlaybackFrameTime
HSequence, 364
GetNextSocket
HTcpServer, 387
GetNvidiaGPUDeviceInfo
HGPUDeviceManager, 147
GetOldestFrameIndex
HSequence, 372
GetOutputDeviceCount
HIODriver, 248
GetOutputDeviceName
HIODriver, 248
GetOutputLevel
HIODriver, 249
GetOutputVolume
HAudioDriverDirectX, 66
GetOverlayColor
HImageOverlay, 230
GetPacketSize
HMetadata, 272
HMetadataInfo, 288
GetPanTilt
HGrabModule, 184
GetPanTiltXRange
HGrabModule, 183
GetPatternOrigin

HImageColorProcessing, 199
GetPeerAddress
HTcpClient, 382
HTcpServer, 388
GetPlaybackFormat
HAviFile, 104
GetPlaybackFrameRate
HAviFile, 104
GetPlaybackFrameSkip
HSequence, 362
GetPlaybackPosition
HDaqFile, 127, 134
HSequence, 360
GetPlaybackRange
HSequence, 363
GetPlaybackRate
HAviFile, 104
GetPlaybackSpeed
HSequence, 361
GetPollingDelay
HIODriver, 247
GetPolygonPixelCount
HImageOverlay, 233
GetPositionAtTime
HDaqFile, 126, 133
GetPositionByTimeStamp
HAACFile, 41
HAudioFile, 72
HMP3File, 328
GetPreAllocatedImages
HSequence, 359
GetProgFolder
HGrabModule, 173
GetROI
HGrabModule, 164
GetRawImageData
HImage, 190
GetReadingPosition
HAACFile, 41
HAudioFile, 72
HDaqFile, 125, 132
HMP3File, 328
GetRecordingPosition
HAudioFile, 72
HDaqFile, 125, 131
HSequence, 356
GetRecordingPositionByRefFrame
HDaqFile, 125, 132
GetRecordingPositionByRefTimeStamp
HDaqFile, 126, 132
GetRedChannel
HImage, 190
GetReferenceFrame
HSequence, 369
GetReferenceInfo
HDaqFile, 126, 133
GetReferenceTime
HSequence, 370
GetRoi
HMotionDetection, 315, 321
GetRotateAngle
HRotation, 343, 347
GetSample
HDaqDataSet, 115, 118
HIODriver, 254
GetSampleCount
HAudioData, 44
HDaqDataSet, 115, 118
GetSampleFactor
HImageColorProcessing, 199
GetSampleRate
HAudioFile, 68
HDaqFile, 125, 132
GetSamplerate
HAudioData, 44
GetSavedProgFolder
HGrabModule, 173
GetSequenceCompression
HSequence, 365
GetSettingsCaps
HGrabModule, 165
GetSettingsCount
HGrabModule, 165
GetSmoothEdge
HRotation, 345, 347
GetSobelDetectionMode
HMotionDetection, 313, 320
GetSocket
HTcpClient, 382
HTcpServer, 386
GetSoftwareName
HAviFile, 106
GetSourceRoi
HImageResize, 240
GetSpecificMetadata
HImage, 193
GetSpecificMetadataCount
HImage, 193
GetTextBackgroundColor
HImageOverlay, 230
GetTextMode
HGrabModule, 172
GetThresholdValue
HMotionDetection, 313, 320
GetTimeAtPosition
HDaqFile, 127, 133
GetTimeStamp
HAACFile, 40
HAudioData, 45
HAudioFile, 72
HDaqDataSet, 115, 118
HMP3File, 327
GetTimeStampImages
HLineScanManager, 265, 268
GetTimeStampMS
HAudioData, 45

HDaqDataSet, 116, 118
GetTimeStampUS
HAudioData, 45
HDaqDataSet, 116, 119
GetTimestamp
HImage, 191
GetTimestampMS
HImage, 192
GetTimestampUS
HImage, 192
GetTitle
HAviFile, 105
GetTotalAudioDatas
HAACFile, 40
HMP3File, 327
HMP4File, 336
GetTotalBufferCount
HGrabModule, 182
GetTrueImageSize
HSequence, 365
GetUID
HMetadata, 271
GetValuesCaps
HGrabModule, 166
GetValuesCount
HGrabModule, 166
GetVideoCodecCount
HAviFile, 98
GetVideoCodecList
HAviExporter, 87
GetVideoCodecName
HAviFile, 98
GetVideoInfo
HMP4File, 334
GetZoom
HGrabModule, 184
HViewer, 398
GetZoomRange
HGrabModule, 183
GrayToMono8
HGrayConverter, 185, 186

H_ABSOFTWARE_ACTIVE_GIGE
HEnums.h, 413
H_ABSOFTWARE_ACTIVE_USB
HEnums.h, 413
H_ACTIVESILICON_FIREBIRD
HEnums.h, 413
H_ACTIVESILICON_PHOENIX
HEnums.h, 413
H_ADVANCE_BOOL_VALUE
HEnums.h, 421
H_ADVANCE_COMMAND
HEnums.h, 421
H_ADVANCE_DOUBLE_VALUE
HEnums.h, 421
H_ADVANCE_ENUMERATION
HEnums.h, 421
H_ADVANCE_INTEGER_VALUE
HEnums.h, 421
H_ADVANCE_OTHERS
HEnums.h, 421
H_AJA_KONA
HEnums.h, 413
H_ANDOR_LUCA
HEnums.h, 413
H_ANDOR_NEO
HEnums.h, 413
H_AUD_FILE_AUD_BLOCK
HEnums.h, 424
H_AUD_FILE_BWAVE_CONTINOUS
HEnums.h, 424
H_AUDIO_COMPRESSION_AAC
HEnums.h, 425
H_AUDIO_COMPRESSION_MP3
HEnums.h, 425
H_AUDIO_COMPRESSION_NONE
HEnums.h, 425
H_AUDIO_FILE_AUD_PCM
HEnums.h, 424
H_AUDIO_FILE_BWAVE_PCM
HEnums.h, 424
H_AUDIO_FLOAT_MSB_PACKED_PCM
HEnums.h, 424
H_AUDIO_FLOAT_MSB_PLANAR_PCM
HEnums.h, 424
H_AUDIO_FLOAT_PACKED_PCM
HEnums.h, 424
H_AUDIO_FLOAT_PLANAR_PCM
HEnums.h, 424
H_AUDIO_INT_MSB_PACKED_PCM
HEnums.h, 424
H_AUDIO_INT_MSB_PLANAR_PCM
HEnums.h, 424
H_AUDIO_INT_PACKED_PCM
HEnums.h, 424
H_AUDIO_INT_PLANAR_PCM
HEnums.h, 424
H_AUDIO_READ_FAILED
HEnums.h, 425
H_AUDIO_SAMPLE_BUFFER_CHANGED
HEnums.h, 425
H_AUDIO_UINT_PACKED_PCM
HEnums.h, 424
H_AUDIO_UINT_PLANAR_PCM
HEnums.h, 424
H_AUDIO_UNKNOWN
HEnums.h, 424
H_AUDIO_UNSUPPORTED_FORMAT
HEnums.h, 425
H_AVI_PLAYBACK_BGR
HEnums.h, 422
H_AVI_PLAYBACK_BGRX
HEnums.h, 422
H_AVI_PLAYBACK_NATIVE
HEnums.h, 422
H_AVT_DCAM

HEnums.h, 413
H_AVT_DCAM_UNI
HEnums.h, 413
H_AVT_UNIVERSAL
HEnums.h, 413
H_AXIS
HEnums.h, 413
H_BASLER_BCAM
HEnums.h, 413
H_BASLER_GIGE
HEnums.h, 413
H_BASLER_IEEE1394
HEnums.h, 413
H_BASLER_PYLON
HEnums.h, 413
H_BASLER_TOF
HEnums.h, 413
H_BAUMER2
HEnums.h, 413
H_BAUMER_FW
HEnums.h, 413
H_BAYER_PATTERNORIGIN_B
HEnums.h, 419
H_BAYER_PATTERNORIGIN_GB
HEnums.h, 419
H_BAYER_PATTERNORIGIN_GR
HEnums.h, 419
H_BAYER_PATTERNORIGIN_R
HEnums.h, 419
H_BAYER_SAMPLEFACTOR_1
HEnums.h, 419
H_BAYER_SAMPLEFACTOR_2
HEnums.h, 419
H_BAYER_SAMPLEFACTOR_4
HEnums.h, 419
H_BAYER_SAMPLEFACTOR_8
HEnums.h, 419
H_BITFLOW_CL
HEnums.h, 413
H_BITFLOW_CXP
HEnums.h, 413
H_BLACK_MAGIC
HEnums.h, 413
H_BYTE_FORMAT
HEnums.h, 419
H_CCAM
HEnums.h, 413
H_CCDDIRECT_MICROPIX
HEnums.h, 413
H_CMU_IEEE1394
HEnums.h, 413
H_COMPRESSION_CAYER
HEnums.h, 418
H_COMPRESSION_H264
HEnums.h, 418
H_COMPRESSION_H264_LOOP
HEnums.h, 418
H_COMPRESSION_H264_RGB
HEnums.h, 418
H_COMPRESSION_H264_RGB_LOOP
HEnums.h, 418
H_COMPRESSION_HEVC
HEnums.h, 418
H_COMPRESSION_HUFFMAN
HEnums.h, 418
H_COMPRESSION_HUFFMAN_FAST
HEnums.h, 418
H_COMPRESSION_JPEG
HEnums.h, 418
H_COMPRESSION_JPEG_LOSSLESS
HEnums.h, 418
H_COMPRESSION_LZ
HEnums.h, 418
H_COMPRESSION_LZ_FAST
HEnums.h, 418
H_COMPRESSION_MPEG4
HEnums.h, 418
H_COMPRESSION_NONE
HEnums.h, 418
H_COMPRESSION_RLE
HEnums.h, 418
H_COMPRESSION_RLE_FAST
HEnums.h, 418
H_COMPRESSION_WAVELET
HEnums.h, 418
H_COOKE_PCO
HEnums.h, 413
H_COOKE_PIXELFLY
HEnums.h, 413
H_COOKE_SENSICAM
HEnums.h, 413
H_COOKE_SENSICAM_2
HEnums.h, 413
H_CORECO_BANDIT2CV
HEnums.h, 413
H_CORECO_BANDIT2MV
HEnums.h, 413
H_CORECO_BANDIT2RGB
HEnums.h, 413
H_CORECO_GIGE
HEnums.h, 413
H_CORECO_PC2CAMLINK
HEnums.h, 413
H_CORECO_PC2COMP
HEnums.h, 413
H_CORECO_PC2VISION
HEnums.h, 413
H_CORECO_PCCAMLINK
HEnums.h, 413
H_CORECO_VIPERDIG
HEnums.h, 413
H_CORECO_X64
HEnums.h, 413
H_CPU_DEVICE_ONLY
HEnums.h, 426
H_CYR_VANILLA

HEnums.h, 426
H_CYR_VANILLA_LOSSY_HI
HEnums.h, 426
H_CYR_VANILLA_LOSSY_LOW
HEnums.h, 426
H_CYR_VANILLA_LOSSY_MED
HEnums.h, 426
H_DAQ_PARAM_ENUM
HEnums.h, 425
H_DAQ_PARAM_FLOAT
HEnums.h, 425
H_DAQ_PARAM_INT
HEnums.h, 425
H_DATAPATH_VISIONRGB
HEnums.h, 414
H_DATETIME_FULL
HEnums.h, 421
H_DATETIME_LTC
HEnums.h, 421
H_DATETIME_STANDARD
HEnums.h, 421
H_DATETIME_TIME_T
HEnums.h, 421
H_EMERGENT
HEnums.h, 414
H_EMS_PHYNXRGB
HEnums.h, 414
H_EMS_XTREMERGB
HEnums.h, 414
H_EMVACAM
HEnums.h, 414
H_EPIPHAN
HEnums.h, 414
H_EPIX_D2X
HEnums.h, 414
H_EPIX_RAPTOR
HEnums.h, 414
H_EPIX_SV5
HEnums.h, 414
H_ERROR_AAC_DECODING
HEnums.h, 431
H_ERROR_ADD_ACODEC
HEnums.h, 428
H_ERROR_ADD_AUDIO_SOURCE
HEnums.h, 428
H_ERROR_ADD_AVISPLITTER
HEnums.h, 429
H_ERROR_ADD_FEEDER
HEnums.h, 428
H_ERROR_ADD_FILEREADER
HEnums.h, 429
H_ERROR_ADD_MUX_FILE
HEnums.h, 428
H_ERROR_ADD_SAMPLE_GRABBER
HEnums.h, 430
H_ERROR_ADD_VCODEC
HEnums.h, 428
H_ERROR_AUDIO_CANTCREATEFILE
HEnums.h, 430
H_ERROR_AUDIO_CHANNEL_MASK_MATCH
HEnums.h, 430
H_ERROR_AUDIO_FORMAT
HEnums.h, 430
H_ERROR_AUDIO_FORMAT_CHANGED
HEnums.h, 430
H_ERROR_AUDIO_INPUTSUPPORT
HEnums.h, 430
H_ERROR_AUDIO_INPUTSUPPORT_LOAD
HEnums.h, 430
H_ERROR_AUDIO_INVALID_FORMAT
HEnums.h, 430
H_ERROR_AUDIO_INVALID_INDEX
HEnums.h, 430
H_ERROR_AUDIO_LOAD
HEnums.h, 430
H_ERROR_AUDIO_NOAUD
HEnums.h, 430
H_ERROR_AUDIO_OUTPUTSUPPORT
HEnums.h, 430
H_ERROR_AUDIO_OUTPUTSUPPORT_LOAD
HEnums.h, 430
H_ERROR_AUDIO_READ_FAILED
HEnums.h, 430
H_ERROR_AUDIO_WRITE_AUD
HEnums.h, 430
H_ERROR_AUIDO_PROCESS_MORE_DATA
HEnums.h, 431
H_ERROR_AUIDO_PROCESS_NO_DATA
HEnums.h, 431
H_ERROR_AVI_TAG_NOT_FOUND
HEnums.h, 431
H_ERROR_BAYER_CONVERSION
HEnums.h, 429
H_ERROR_BIND_SOCKET
HEnums.h, 430
H_ERROR_BUFFER_INTERNAL
HEnums.h, 427
H_ERROR_CANT_ADD_FRAME
HEnums.h, 430
H_ERROR_CANTCREATEFILE
HEnums.h, 429
H_ERROR_CLIENT_ALREADY_CONNECTED
HEnums.h, 430
H_ERROR_CLOSE_SOCKET
HEnums.h, 430
H_ERROR_COLOR_BALANCING
HEnums.h, 429
H_ERROR_CONNECT_ACODEC_MUX
HEnums.h, 428
H_ERROR_CONNECT_ALEVEL_ACODEC
HEnums.h, 430
H_ERROR_CONNECT_ALEVEL_MUX
HEnums.h, 430
H_ERROR_CONNECT_ASOURCE_ACODEC
HEnums.h, 428
H_ERROR_CONNECT_ASOURCE_ALEVEL

HEnums.h, 430
H_ERROR_CONNECT_ASOURCE_MUX
HEnums.h, 428
H_ERROR_CONNECT_SOURCE_SPLITTER
HEnums.h, 428
H_ERROR_CONNECT_SPLITTER_RENDERER
HEnums.h, 428
H_ERROR_CONNECT_VCODEC_MUX
HEnums.h, 428
H_ERROR_CONNECT_VSOURCE_MUX
HEnums.h, 428
H_ERROR_CONNECT_VSOURCE_VCODEC
HEnums.h, 428
H_ERROR_CONNECTION_FAILED
HEnums.h, 430
H_ERROR_CONVERT_BGR24
HEnums.h, 427
H_ERROR_CONVERT_BGRX
HEnums.h, 428
H_ERROR_CONVERT_UNSUPPORTED
HEnums.h, 428
H_ERROR_CONVERT_UVY411
HEnums.h, 428
H_ERROR_CONVERT_UVY422
HEnums.h, 428
H_ERROR_CONVERT_UVY444
HEnums.h, 428
H_ERROR_CONVERT_YUV422
HEnums.h, 428
H_ERROR_CREATE_BUFFER
HEnums.h, 427
H_ERROR_CREATE_CAPTURE_BUILDER
HEnums.h, 428
H_ERROR_CREATE_CLASS_ENUM
HEnums.h, 428
H_ERROR_CREATE_FEEDER
HEnums.h, 428
H_ERROR_CREATE_FILE_ERROR
HEnums.h, 431
H_ERROR_CREATE_GRAPH_BUILDER
HEnums.h, 428
H_ERROR_CREATE_RENDER
HEnums.h, 429
H_ERROR_CREATE_SAMPLE_GRABBER
HEnums.h, 430
H_ERROR_CREATE_SOCKET
HEnums.h, 430
H_ERROR_CREATE_SYSTEM_ENUM
HEnums.h, 428
H_ERROR_CREATEC_UNSUPPORT_FORMAT
HEnums.h, 431
H_ERROR_DAQSUPPORT_CANTCREATEFILE
HEnums.h, 429
H_ERROR_DAQSUPPORT_FORMAT_CHANGED
HEnums.h, 430
H_ERROR_DAQSUPPORT_INVALID_INDEX
HEnums.h, 430
H_ERROR_DAQSUPPORT_LOAD
HEnums.h, 429
H_ERROR_DAQSUPPORT_NODAQ
HEnums.h, 430
H_ERROR_DAQSUPPORT_READ_FAILED
HEnums.h, 429
H_ERROR_DAQSUPPORT_WRITE_DAQ
HEnums.h, 430
H_ERROR_DECODE_MORE_DATA
HEnums.h, 431
H_ERROR_DONTSUPPORTREWRITE
HEnums.h, 429
H_ERROR_EXPORT_FAILED
HEnums.h, 429
H_ERROR_EXPORT_IN_PROGRESS
HEnums.h, 427
H_ERROR_FAILED_TO_ADD_TAG
HEnums.h, 431
H_ERROR_FAILEDREGISTER
HEnums.h, 427
H_ERROR_FILENOTFOUND
HEnums.h, 429
H_ERROR_FRAME_SKIPPED
HEnums.h, 431
H_ERROR_GDIDRAW
HEnums.h, 429
H_ERROR_GPU_IMAGE_PUSH
HEnums.h, 431
H_ERROR_H264_DECODING
HEnums.h, 430
H_ERROR_H264_ENCODING
HEnums.h, 430
H_ERROR_H264_UNSUPPOT_IMAGE_SIZE
HEnums.h, 430
H_ERROR_HEVC_DECODING
HEnums.h, 431
H_ERROR_HEVC_ENCODING
HEnums.h, 431
H_ERROR_IMAGE_FORMAT
HEnums.h, 429
H_ERROR_IMAGEFORMATCHANGED
HEnums.h, 429
H_ERROR_INI_FAIL
HEnums.h, 427
H_ERROR_INPUTSUPPORT_DEV_INIT_FAILED
HEnums.h, 429
H_ERROR_INPUTSUPPORT_DEV_UNINIT_FAILED
HEnums.h, 429
H_ERROR_INPUTSUPPORT_LOAD
HEnums.h, 429
H_ERROR_INPUTSUPPORT_MONITOR
HEnums.h, 429
H_ERROR_INTERLEAVE_MODE
HEnums.h, 428
H_ERROR_INTERLEAVE_TIME
HEnums.h, 428
H_ERROR_INVALID_COMPRESSION
HEnums.h, 427
H_ERROR_INVALID_DIGIT

HEnums.h, 427
H_ERROR_INVALID_INDEX
HEnums.h, 429
H_ERROR_INVALID_LUT_FILE
HEnums.h, 431
H_ERROR_INVALID_SOCKET
HEnums.h, 430
H_ERROR_LISTEN_SOCKET
HEnums.h, 430
H_ERROR_LOADING_AUDIO_CODEC
HEnums.h, 428
H_ERROR_LOADING_AUDIO_SOURCE
HEnums.h, 428
H_ERROR_LOADING_VIDEO_CODEC
HEnums.h, 428
H_ERROR_MALFORMED_XML_FILE
HEnums.h, 431
H_ERROR_MERGE_ERROR
HEnums.h, 429
H_ERROR_MIRROR_ERROR
HEnums.h, 429
H_ERROR_MONIKER_NULL
HEnums.h, 428
H_ERROR_MP3_DECODING
HEnums.h, 431
H_ERROR_MP4_AUDIO_READ
HEnums.h, 431
H_ERROR_MP4_VIDEO_READ
HEnums.h, 431
H_ERROR_MPEG4_DECODING
HEnums.h, 431
H_ERROR_MPEG4_ENCODING
HEnums.h, 431
H_ERROR_NO_AUDIO_SOURCE
HEnums.h, 430
H_ERROR_NO_BEGINEXPORT
HEnums.h, 427
H_ERROR_NO_CAPTURE_GRAPH
HEnums.h, 431
H_ERROR_NO_CODEC_SETTINGS
HEnums.h, 428
H_ERROR_NO_FILE
HEnums.h, 430
H_ERROR_NO_GRABBER
HEnums.h, 427
H_ERROR_NO_GRABMAN
HEnums.h, 427
H_ERROR_NO_IMAGE_DATA
HEnums.h, 427
H_ERROR_NO_PROCESSING_APPLIED
HEnums.h, 429
H_ERROR_NO_RECORDING
HEnums.h, 430
H_ERROR_NOAUTHORIZATION
HEnums.h, 427
H_ERROR_NOBITMAP
HEnums.h, 429
H_ERROR_NOCOMPORT
HEnums.h, 429
H_ERROR_NOSEQ
HEnums.h, 428
H_ERROR_NOT_CONNECTED
HEnums.h, 431
H_ERROR_OUT_OF_RANGE
HEnums.h, 430
H_ERROR_OUTOFMEMORY
HEnums.h, 429
H_ERROR_OUTPUTSUPPORT_DEV_INIT_FAILED
HEnums.h, 429
H_ERROR_OUTPUTSUPPORT_DEV_UNINIT_FAIL↩
ED
HEnums.h, 429
H_ERROR_OUTPUTSUPPORT_LOAD
HEnums.h, 429
H_ERROR_QUICKTIME_ERROR
HEnums.h, 429
H_ERROR_QUICKTIME_INIFAIL
HEnums.h, 429
H_ERROR_REGISTRY_READ
HEnums.h, 431
H_ERROR_REGISTRY_WRITE
HEnums.h, 431
H_ERROR_RESIZE_ERROR
HEnums.h, 429
H_ERROR_ROTATION_ERROR
HEnums.h, 429
H_ERROR_SAMPLE_GRABBER_CALLBACK
HEnums.h, 430
H_ERROR_SCRIPTER_NOT_RUNNING
HEnums.h, 429
H_ERROR_SCRIPTRUNNING
HEnums.h, 429
H_ERROR_SECTORBYTES_ZERO
HEnums.h, 429
H_ERROR_SEND
HEnums.h, 430
H_ERROR_SEQ_CANTCREATEFILE
HEnums.h, 428
H_ERROR_SEQ_INVALIDDATA
HEnums.h, 428
H_ERROR_SEQ_INVALIDDEPTH
HEnums.h, 428
H_ERROR_SEQ_INVALIDFRAMERATE
HEnums.h, 428
H_ERROR_SEQ_INVALIDHEIGHT
HEnums.h, 428
H_ERROR_SEQ_INVALIDIMAGESIZE
HEnums.h, 428
H_ERROR_SEQ_INVALIDLENGTH
HEnums.h, 428
H_ERROR_SEQ_INVALIDMAGICNUMBER
HEnums.h, 428
H_ERROR_SEQ_INVALIDWIDTH
HEnums.h, 428
H_ERROR_SEQ_UNSUPPORTED_FORMAT
HEnums.h, 428

H_ERROR_SEQ_WRONGHEADERSIZE
HEnums.h, 428
H_ERROR_SERVER_ALREADY_OPEN
HEnums.h, 430
H_ERROR_SERVER_NOT_RUNNING
HEnums.h, 430
H_ERROR_SHUTDOWN_SOCKET
HEnums.h, 430
H_ERROR_SOCKET_ALREADY_CREATED
HEnums.h, 431
H_ERROR_STOP_PLAYBACK
HEnums.h, 429
H_ERROR_STREAMING
HEnums.h, 429
H_ERROR_SUCCESS
HEnums.h, 427
H_ERROR_SYNCH_SOURCE
HEnums.h, 428
H_ERROR_TOOMUCHDATA
HEnums.h, 429
H_ERROR_UNSUPPORT_FUNCTION
HEnums.h, 430
H_ERROR_UNSUPPORTED
HEnums.h, 429
H_ERROR_UNSUPPORTED_ADJUSTMENTS
HEnums.h, 429
H_ERROR_UNSUPPORTED_ADVANCEFEATURE
HEnums.h, 429
H_ERROR_UNSUPPORTED_INTERFACE
HEnums.h, 431
H_ERROR_UNSUPPORTED_SETTINGS
HEnums.h, 429
H_ERROR_VIDEOOUTPUTSUPPORT_LOAD
HEnums.h, 429
H_ERROR_WRITE_ERROR
HEnums.h, 431
H_ERROR_WRITE_SEQ
HEnums.h, 428
H_ERROR_WRONG_MODE
HEnums.h, 429
H_EURESYS_GRABLINKVALUE
HEnums.h, 414
H_EURESYS_IOTA
HEnums.h, 414
H_EURESYS_PICOLO
HEnums.h, 414
H_EURESYS_PICOLO_HD
HEnums.h, 414
H_EXPORT_AVI
HEnums.h, 425
H_EXPORT_BMP
HEnums.h, 425
H_EXPORT_DPX
HEnums.h, 425
H_EXPORT_FIT
HEnums.h, 425
H_EXPORT_JP2
HEnums.h, 425
H_EXPORT_JPG
HEnums.h, 425
H_EXPORT_MOV
HEnums.h, 425
H_EXPORT_PNG
HEnums.h, 425
H_EXPORT_SEQ
HEnums.h, 425
H_EXPORT_TIFF
HEnums.h, 425
H_FAST_CORRECTION_AUTO
HEnums.h, 421
H_FAST_CORRECTION_MANUAL
HEnums.h, 421
H_FAST_CORRECTION_NONE
HEnums.h, 421
H_FCAV_ARINC818
HEnums.h, 414
H_FOCULUS_CAMERA
HEnums.h, 414
H_FORESIGHTIMAGING_ACCUSTREAM
HEnums.h, 414
H_FR_STREAM_USB
HEnums.h, 414
H_GEVICAM
HEnums.h, 414
H_GIGALINX
HEnums.h, 414
H_GIGALINX_SONY
HEnums.h, 414
H_GPU_DEVICE_AMD
HEnums.h, 426
H_GPU_DEVICE_INTEL
HEnums.h, 426
H_GPU_DEVICE_NVIDIA_CUDA
HEnums.h, 426
H_GPU_DEVICE_NVIDIA_OPENCL
HEnums.h, 426
H_GPU_DEVICE_UNKNOWN
HEnums.h, 426
H_GRABBER_INFO_TYPE_DOUBLE
HEnums.h, 417
H_GRABBER_INFO_TYPE_LONG
HEnums.h, 417
H_GRABBER_INFO_TYPE_STRING
HEnums.h, 417
H_GRABBER_NOTIFICATION_CAPTURE_ERROR
HEnums.h, 423
H_GRABBER_NOTIFICATION_CAPTURE_EVENT_↩
RECEIVED
HEnums.h, 423
H_GRABBER_NOTIFICATION_CONNECTED
HEnums.h, 423
H_GRABBER_NOTIFICATION_CONNECTION_LOST
HEnums.h, 423
H_GRABBER_NOTIFICATION_DISCONNECTED
HEnums.h, 423
H_GRABBER_NOTIFICATION_FRAME_DROP

HEnums.h, 423
H_GRABBER_NOTIFICATION_INVALID_SIGNAL
HEnums.h, 423
H_GRABBER_NOTIFICATION_PTP_CLOCK_STAT↩US
HEnums.h, 423
H_GRABBER_NOTIFICATION_SIGNAL_PRESENT
HEnums.h, 423
H_GRABBER_NOTIFICATION_TIMESTAP_SET
HEnums.h, 423
H_GRT_MATRIX
HEnums.h, 414
H_GRT_MATRIX_HL2V
HEnums.h, 414
H_HAMAMATSU_DCAM
HEnums.h, 414
H_HASOTEC_FG30
HEnums.h, 414
H_HITACHI_DCAM
HEnums.h, 414
H_HL2V
HEnums.h, 414
H_IDS_UEYE
HEnums.h, 414
H_IMAGE_BGR
HEnums.h, 417
H_IMAGE_BGR10_PPACKED
HEnums.h, 418
H_IMAGE_BGR10_PPACKED_PHOENIX
HEnums.h, 418
H_IMAGE_BGR555_PACKED
HEnums.h, 417
H_IMAGE_BGR565_PACKED
HEnums.h, 417
H_IMAGE_BGRx
HEnums.h, 417
H_IMAGE_GRT_DUAL_IMAGE_RGB
HEnums.h, 418
H_IMAGE_GVSP_BGR10V1_PACKED
HEnums.h, 418
H_IMAGE_GVSP_BGR10V2_PACKED
HEnums.h, 418
H_IMAGE_GVSP_PIX_MONO10_BAYER_PACKED
HEnums.h, 418
H_IMAGE_GVSP_PIX_MONO10_PACKED
HEnums.h, 418
H_IMAGE_MONO
HEnums.h, 417
H_IMAGE_MONO80_64LSB_16MSB
HEnums.h, 418
H_IMAGE_MONO_BAYER
HEnums.h, 417
H_IMAGE_MONO_BAYER_MSB
HEnums.h, 417
H_IMAGE_MONO_BAYER_MSB_SWAP
HEnums.h, 417
H_IMAGE_MONO_BAYER_PPACKED
HEnums.h, 418
H_IMAGE_MONO_BAYER_PPACKED_8448
HEnums.h, 418
H_IMAGE_MONO_BAYER_PPACKED_8844
HEnums.h, 418
H_IMAGE_MONO_MSB
HEnums.h, 417
H_IMAGE_MONO_MSB_SWAP
HEnums.h, 417
H_IMAGE_MONO_PPACKED
HEnums.h, 418
H_IMAGE_MONO_PPACKED_8448
HEnums.h, 418
H_IMAGE_MONO_PPACKED_8844
HEnums.h, 418
H_IMAGE_PHOTON_FOCUS_DR_BAYER
HEnums.h, 418
H_IMAGE_PHOTON_FOCUS_DR_MONO
HEnums.h, 418
H_IMAGE_PHOTON_FOCUS_QR_BAYER
HEnums.h, 418
H_IMAGE_PHOTON_FOCUS_QR_MONO
HEnums.h, 418
H_IMAGE_PLANAR
HEnums.h, 417
H_IMAGE_RGB
HEnums.h, 417
H_IMAGE_RGB10_PPACKED_PHOENIX
HEnums.h, 418
H_IMAGE_UNKNOWN
HEnums.h, 417
H_IMAGE_UVY411
HEnums.h, 417
H_IMAGE_UVY422
HEnums.h, 417
H_IMAGE_UVY444
HEnums.h, 417
H_IMAGE_YUV420_PLANAR
HEnums.h, 417
H_IMAGE_YUV422
HEnums.h, 417
H_IMAGINGSOURCE
HEnums.h, 414
H_IMITECH_1394
HEnums.h, 414
H_IMITECH_GIGE
HEnums.h, 414
H_IMITECH_NEPTUNE
HEnums.h, 414
H_IMPERX_BOBCAT_GIGE
HEnums.h, 414
H_IMPERX_FRAMELINK
HEnums.h, 414
H_IMPERX_FRAMELINK_EXPRESS
HEnums.h, 414
H_IMPERX_GIGE
HEnums.h, 414
H_IMPERX_SDI_EXPRESS
HEnums.h, 414

H_IMPERX_VCE_EXPRESS
HEnums.h, 414
H_IMPERX_VCEPRO
HEnums.h, 414
H_INPUT_EVENT_FALLING_EDGE
HEnums.h, 420
H_INPUT_EVENT_RISING_EDGE
HEnums.h, 420
H_INPUT_LEVEL_HIGH
HEnums.h, 420
H_INPUT_LEVEL_LOW
HEnums.h, 420
H_INPUT_LEVEL_UNDEFINED
HEnums.h, 420
H_INTEGRAL_FLASHBUSDX
HEnums.h, 414
H_INTEGRAL_FLASHBUSMV
HEnums.h, 414
H_INTEGRAL_FLASHBUSMX
HEnums.h, 414
H_INTEGRAL_SPECTRIM
HEnums.h, 414
H_IP_CAMERA
HEnums.h, 414
H_ISG_1394
HEnums.h, 415
H_JAIGENICAM2
HEnums.h, 415
H_JAIGENICAM_GIGE
HEnums.h, 415
H_JAIPULNIX_GIGE
HEnums.h, 415
H_JENOPTIK_DIJ
HEnums.h, 415
H_JENOPTIK_PROGRESC10
HEnums.h, 415
H_JENOPTIK_PROGRESC14
HEnums.h, 415
H_JENOPTIK_PROGRESCXX
HEnums.h, 415
H_JPEG_ACCURATE
HEnums.h, 417
H_JPEG_DE_ALGO_GPU_INTEL
HEnums.h, 427
H_JPEG_DE_ALGO_IPP
HEnums.h, 427
H_JPEG_DE_ALGO_TURBO
HEnums.h, 427
H_JPEG_DEFAULT
HEnums.h, 417
H_JPEG_EN_ALGO_GPU
HEnums.h, 427
H_JPEG_EN_ALGO_IPP
HEnums.h, 427
H_JPEG_EN_ALGO_TURBO
HEnums.h, 427
H_JPEG_FAST
HEnums.h, 417
H_JPEG_IMAGE_EXPORT_FREE_IMAGE
HEnums.h, 427
H_JPEG_IMAGE_EXPORT_GPU
HEnums.h, 427
H_JPEG_IMAGE_EXPORT_IPP
HEnums.h, 427
H_JPEG_IMAGE_EXPORT_TURBO
HEnums.h, 427
H_JPEG_QUALITYAVERAGE
HEnums.h, 417
H_JPEG_QUALITYBAD
HEnums.h, 417
H_JPEG_QUALITYGOOD
HEnums.h, 417
H_JPEG_QUALITYNORMAL
HEnums.h, 417
H_JPEG_QUALITYSUPERB
HEnums.h, 417
H_KAPPA_FW
HEnums.h, 415
H_KAPPA_HIRES3XR
HEnums.h, 415
H_KAYA
HEnums.h, 415
H_KINECT
HEnums.h, 415
H_LEUTRON_PICSIGHT_GIGE
HEnums.h, 415
H_LEUTRON_SIMPLON_GIGE
HEnums.h, 415
H_LUMENERA_GIGE
HEnums.h, 415
H_LUMENERA_USB2
HEnums.h, 415
H_MATRIXVISION
HEnums.h, 415
H_MATRIXVISION_BLUECOUGAR
HEnums.h, 415
H_MATRIXVISION_BLUEFOX
HEnums.h, 415
H_MATRIXVISION_HYPERION
HEnums.h, 415
H_MATRIXVISION_SIGMA
HEnums.h, 415
H_MATRIXVISION_TITAN
HEnums.h, 415
H_MATROX_CL
HEnums.h, 415
H_MATROX_CXP
HEnums.h, 415
H_MATROX_HELIOS
HEnums.h, 415
H_MATROX_METEOR2_CL
HEnums.h, 415
H_MATROX_METEOR2_DIG
HEnums.h, 415
H_MATROX_METEOR2_MC
HEnums.h, 415

H_MATROX_METEOR2_STD
HEnums.h, 415
H_MATROX_MORPHIS
HEnums.h, 415
H_MATROX_ODYSSEY
HEnums.h, 415
H_MATROX_ORION
HEnums.h, 415
H_MATROX_RADIENT
HEnums.h, 415
H_MATROX_SOLIOS_XCL
HEnums.h, 415
H_MATROX_VIO_XCL
HEnums.h, 415
H_MBCS_FORMAT
HEnums.h, 419
H_MERGE_HORZ
HEnums.h, 424
H_MERGE_VERT
HEnums.h, 424
H_METADATA_BINARY
HEnums.h, 426
H_METADATA_BOOL
HEnums.h, 426
H_METADATA_BYTE
HEnums.h, 426
H_METADATA_DOUBLE
HEnums.h, 426
H_METADATA_INT
HEnums.h, 426
H_METADATA_INT64
HEnums.h, 426
H_METADATA_SHORT
HEnums.h, 426
H_METADATA_STRING
HEnums.h, 426
H_METADATA_UINT
HEnums.h, 426
H_METADATA_UINT64
HEnums.h, 426
H_METADATA_UNKNOWN
HEnums.h, 426
H_METADATA_USHORT
HEnums.h, 426
H_MIGHTEX
HEnums.h, 415
H_MIGHTEX_CCD
HEnums.h, 415
H_MIRROR_BOTH
HEnums.h, 424
H_MIRROR_HORZ
HEnums.h, 424
H_MIRROR_VERT
HEnums.h, 424
H_MONO_TO_PSEUDO_COLOR_DISABLED
HEnums.h, 423
H_MONO_TO_PSEUDO_COLOR_LUT_COLD
HEnums.h, 423
H_MONO_TO_PSEUDO_COLOR_LUT_HOT
HEnums.h, 423
H_MONO_TO_PSEUDO_COLOR_LUT_INVERTED↩
_RAINBOW
HEnums.h, 423
H_MONO_TO_PSEUDO_COLOR_LUT_NEGATIVE
HEnums.h, 423
H_MONO_TO_PSEUDO_COLOR_LUT_RAINBOW
HEnums.h, 423
H_MONO_TO_PSEUDO_COLOR_LUT_SATURATI↩
ON_3
HEnums.h, 423
H_MONO_TO_PSEUDO_COLOR_LUT_SATURATI↩
ON_5
HEnums.h, 423
H_MOTION_DETECTION_ABS_DIFF
HEnums.h, 422
H_MOTION_DETECTION_EDGE
HEnums.h, 422
H_MOTION_DETECTION_EDGE_ADAPTIVE
HEnums.h, 422
H_MOTION_DETECTION_MAX_ABS_DIFF
HEnums.h, 422
H_MOTION_DETECTION_MEAN
HEnums.h, 422
H_MOVIE_PLAYBACK
HEnums.h, 421
H_MOVIE_RECORDING
HEnums.h, 421
H_MOVIE_UNDEFINED
HEnums.h, 421
H_NET_GIGE
HEnums.h, 415
H_NET_ICUBE
HEnums.h, 415
H_NI_GRABBERS_0
HEnums.h, 415
H_NIKON_D7000
HEnums.h, 415
H_NORPIX_VIRTUAL
HEnums.h, 415
H_OLYMPUS_DP70
HEnums.h, 415
H_ONVIF
HEnums.h, 415
H_OPTRONICS_MACROFIRE
HEnums.h, 415
H_OPTRONICS_MAGNAFIRE
HEnums.h, 416
H_OPTRONICS_MICROFIRE
HEnums.h, 416
H_OUTPUT_LEVEL_HIGH
HEnums.h, 420
H_OUTPUT_LEVEL_LOW
HEnums.h, 420
H_OUTPUT_LEVEL_UNDEFINED
HEnums.h, 420
H_PICOLO_H264_UX

HEnums.h, 414
H_PICOLO_H264_V16
HEnums.h, 414
H_PIXELINK_PLA500
HEnums.h, 416
H_PIXELINK_PLA600
HEnums.h, 416
H_PIXELINK_PLA700
HEnums.h, 416
H_PIXELINK_TM1020
HEnums.h, 416
H_PIXELINK_VCA1281
HEnums.h, 416
H_PIXERA_PENGUIN
HEnums.h, 416
H_PLEORA_EBUS
HEnums.h, 416
H_PLEORA_PT1000CL
HEnums.h, 416
H_PRINCETON
HEnums.h, 416
H_PROSILICA
HEnums.h, 416
H_PROSILICA_GIGE
HEnums.h, 416
H_PROTECTION_INVALID
HEnums.h, 419
H_PROTECTION_NONE
HEnums.h, 419
H_PROTECTION_VALID
HEnums.h, 419
H_PTGREY
HEnums.h, 416
H_PTGREY_LADYBUG
HEnums.h, 416
H_QIMAGING_MP
HEnums.h, 416
H_QIMAGING_RETIGA
HEnums.h, 416
H_REDLAKE_MEGAPLUSII
HEnums.h, 416
H_RESIZE_INTER_CUBIC
HEnums.h, 422
H_RESIZE_INTER_LANCZOS
HEnums.h, 422
H_RESIZE_INTER_LINEAR
HEnums.h, 422
H_RESIZE_INTER_NN
HEnums.h, 422
H_RESIZE_INTER_SUPER
HEnums.h, 422
H_ROPER_CASCADE
HEnums.h, 416
H_ROPER_COOLSNAPCF
HEnums.h, 416
H_ROTATION_INTER_CUBIC
HEnums.h, 422
H_ROTATION_INTER_LINEAR
HEnums.h, 422
H_ROTATION_INTER_NN
HEnums.h, 422
H_SBF_FIREWIRE
HEnums.h, 416
H_SCION_CFW
HEnums.h, 416
H_SILICONSOFTWARE_MICROENABLE
HEnums.h, 416
H_SILICONSOFTWARE_MICROENABLE5
HEnums.h, 416
H_SILLICONIMAGING_GIGE
HEnums.h, 416
H_SOBEL_BOTH
HEnums.h, 422
H_SOBEL_HORIZONTAL
HEnums.h, 422
H_SOBEL_VERTICAL
HEnums.h, 422
H_SPECIAL_COPY_BGR555
HEnums.h, 420
H_SPECIAL_COPY_BGR565
HEnums.h, 420
H_SPECIAL_COPY_BGRA
HEnums.h, 420
H_SPECIAL_COPY_BYR1
HEnums.h, 420
H_SPECIAL_COPY_BYR2
HEnums.h, 420
H_SPECIAL_COPY_BYR4
HEnums.h, 420
H_SPECIAL_COPY_H264
HEnums.h, 420
H_SPECIAL_COPY_JPEG
HEnums.h, 420
H_SPECIAL_COPY_RG30
HEnums.h, 420
H_SPECIAL_COPY_RG48
HEnums.h, 420
H_SPECIAL_COPY_UVY422
HEnums.h, 420
H_SPECIAL_COPY_Y16
HEnums.h, 420
H_SPECIAL_COPY_Y8
HEnums.h, 420
H_SPECIAL_COPY_YUV411
HEnums.h, 420
H_SPECIAL_COPY_YUV422
HEnums.h, 420
H_SPECIAL_COPY_YUV444
HEnums.h, 420
H_SPINNAKER
HEnums.h, 416
H_SPOT
HEnums.h, 416
H_SUGGESTED_ACTION_DISABLE_FRAME_DROP
HEnums.h, 423
H_SUGGESTED_ACTION_ENABLE_FRAME_DROP

HEnums.h, 423
H_SUGGESTED_ACTION_NONE
HEnums.h, 423
H_SUGGESTED_ACTION_OPEN_HARDWARESET↩
TINGDLG
HEnums.h, 423
H_SUGGESTED_ACTION_PAUSE_STREAM
HEnums.h, 423
H_SUGGESTED_ACTION_RELOAD_GRABBER
HEnums.h, 423
H_SUGGESTED_ACTION_RESTART_STREAM
HEnums.h, 423
H_SUGGESTED_ACTION_RESUME_STREAM
HEnums.h, 423
H_SUGGESTED_ACTION_UNLOAD_GRABBER
HEnums.h, 423
H_SUMIX_12A2
HEnums.h, 416
H_SUMIX_150
HEnums.h, 416
H_SUMIX_15M5X
HEnums.h, 416
H_SUMIX_16EXX
HEnums.h, 416
H_SUMIX_M7X
HEnums.h, 416
H_SUMIX_M8X
HEnums.h, 416
H_SVS_VISTEK
HEnums.h, 416
H_SVS_VISTEK_GIGE
HEnums.h, 416
H_SYNC_STATUS_FREERUN
HEnums.h, 419
H_SYNC_STATUS_HOLDOVER
HEnums.h, 419
H_SYNC_STATUS_SYNCHRONIZED
HEnums.h, 419
H_SYNC_STATUS_UNKOWN
HEnums.h, 419
H_SYNC_STATUS_UNSUPPORTED
HEnums.h, 419
H_TATTILE_DUPLEX
HEnums.h, 416
H_TS_PARAM_ENUM
HEnums.h, 426
H_TS_PARAM_FLOAT
HEnums.h, 426
H_TS_PARAM_INT
HEnums.h, 426
H_UNICODE_FORMAT
HEnums.h, 419
H_VIEWORKS
HEnums.h, 416
H_VIMBA
HEnums.h, 416
H_VRMAGIC_RIC10
HEnums.h, 416
H_WDM_CAMERA
HEnums.h, 416
H_XCELERA
HEnums.h, 416
H_XENICS
HEnums.h, 416
H_XIMEA
HEnums.h, 416
HAACFile, 37
AddData, 39
Close, 41
Create, 39
EndRecording, 39
GetAudioInfo, 40
GetLastError, 39
GetPositionByTimeStamp, 41
GetReadingPosition, 41
GetTimeStamp, 40
GetTotalAudioDatas, 40
Open, 40
ReadAudio, 41
SetAudioParam, 39
SetErrorOutput, 38
SetReadingPosition, 41
HAudioData, 42
CloneAudioData, 43
CreateAudioData, 43
GetAudioBitDepth, 45
GetAudioBitDepthReal, 45
GetAudioDataBuffer, 44
GetChannelNum, 44
GetFormat, 44
GetSampleCount, 44
GetSamplerate, 44
GetTimeStamp, 45
GetTimeStampMS, 45
GetTimeStampUS, 45
ReallocateAudio, 44
SetTimeStamp, 45
HAudioDriverASIO, 46
AudioInputPresent, 48
AudioOutputPresent, 52
CloseAudioInputDevice, 49
CloseAudioOutputDevice, 53
GetAudioInputChannelCount, 50
GetAudioInputChannelInfoByChannelIndex, 51
GetAudioInputDeviceCount, 48
GetAudioInputDeviceName, 48
GetAudioInputSampleRate, 50
GetAudioInputSampleRateInfo, 50
GetAudioInputSampleRateInfoCount, 49
GetAudioOutputChannelCount, 54
GetAudioOutputChannelInfoByChannelIndex, 55
GetAudioOutputDeviceCount, 52
GetAudioOutputDeviceName, 52
GetAudioOutputSamplePerBuffer, 53
GetAudioOutputSampleRate, 54
GetAudioOutputSampleRateInfo, 54

GetAudioOutputSampleRateInfoCount, 53
GetCaptureBufferUsage, 52
GetDriverName, 48
GetLastError, 48
HAudioDriverASIO, 47
IsAudioInputCapturing, 52
IsAudioOutputPlaying, 56
OpenAudioInputDevice, 49
OpenAudioInputHardwareSettingsDlg, 50
OpenAudioOutputDevice, 53
OpenAudioOutputHardwareSettingsDlg, 54
ResetCaptureBufferUsage, 52
SetAudioInputTimeSourceDLL, 49
SetCaptureBufferCount, 52
SetErrorOutput, 48
StartAudioInputCapture, 51
StartAudioOutputPlayback, 55
StopAudioInputCapture, 51
StopAudioOutputPlayback, 55
HAudioDriverDirectX, 56
AudioInputPresent, 58
AudioOutputPresent, 62
CloseAudioInputDevice, 59
CloseAudioOutputDevice, 63
GetAudioInputDeviceCount, 58
GetAudioInputDeviceName, 59
GetAudioInputFormat, 61
GetAudioInputFormatCount, 60
GetAudioInputFormatInfo, 60
GetAudioOutputDeviceCount, 62
GetAudioOutputDeviceName, 62
GetAudioOutputFormat, 64
GetAudioOutputFormatCount, 63
GetAudioOutputFormatInfo, 64
GetAudioOutputSampleBufferSize, 63
GetCaptureBufferUsage, 62
GetDriverName, 58
GetLastError, 58
GetOutputVolume, 66
HAudioDriverDirectX, 58
IsAudioInputCapturing, 61
IsAudioOutputPlaying, 65
OpenAudioInputDevice, 59
OpenAudioOutputDevice, 63
ResetCaptureBufferUsage, 62
SetAudioInputFormat, 60
SetAudioInputTimeSourceDLL, 59
SetAudioOutputFormat, 64
SetCaptureBufferCount, 62
SetErrorOutput, 58
SetOutputVolume, 65
StartAudioInputCapture, 61
StartAudioOutputPlayback, 65
StopAudioInputCapture, 61
StopAudioOutputPlayback, 65
HAudioFile, 66
Close, 70
Create, 69
GetAllocatedAudioDatas, 71
GetAudioChannelCount, 68
GetAudioDepth, 68
GetAudioDepthReal, 68
GetAudioFormat, 68
GetAudioSamplesPerBuffer, 68
GetAufioFileType, 69
GetLastError, 68
GetPositionByTimeStamp, 72
GetReadingPosition, 72
GetRecordingPosition, 72
GetSampleRate, 68
GetTimeStamp, 72
Open, 69
PreparePlayback, 70
Read, 70, 71
SetErrorOutput, 68
SetPositionByTimeStamp, 73
SetReadingPosition, 71
SetStartTimeStamp, 72
Write, 70
HAudioPlayer, 73
GetLastError, 74
IsPlaying, 78
MoveTo, 79
OpenPlayer, 77, 78
Play, 78
PushLiveAudioData, 78
SetErrorOutput, 74
SetFileSource, 74, 76
SetLiveSource, 76
Stop, 78
HAudioRecorder, 79
AreChannelsOpen, 84
CloseChannels, 83
EnablePassthrough, 85
GetAudioInputLevel, 84
GetBufferCount, 85
GetBufferUsage, 85
GetLastError, 80
GetMaxBufferUsage, 85
IsRecording, 84
OpenChannels, 83
SetAudioDriver, 81
SetAudioFile, 82, 83
SetBufferCount, 85
SetCallback, 81
SetErrorOutput, 80
StartRecording, 84
StopRecording, 84
HAutoLocker, 86
HAviExporter, 86
BeginExport, 90
EndExport, 90
ExportImage, 90
ForceFrameRate, 90
GetAudioCodecList, 89
GetAudioSourceList, 88

GetLastError, 87
GetVideoCodecList, 87
IsExporting, 90
SetAudioCodec, 89
SetAudioSource, 88
SetErrorOutput, 87
SetVideoCodec, 88
ShowAudioCodecSettings, 89
ShowVideoCodecSettings, 88
UseSpecialCodec, 89
HAviFile, 91
AddFrame, 97
CloseAviFile, 97
CreateAviFile, 95, 96
DetectVideoCodecType, 100
ForceFrameRate, 101
GetAudioCodecCount, 99
GetAudioCodecName, 99
GetAudioSourceCount, 98
GetAudioSourceName, 99
GetAuthor, 106
GetAviMode, 97
GetCopyright, 105
GetCreationDateTime, 105
GetCurrentPosition, 103
GetDateTimeOriginal, 106
GetFrameCount, 103
GetIndexAt, 103
GetLastError, 94
GetPlaybackFormat, 104
GetPlaybackFrameRate, 104
GetPlaybackRate, 104
GetSoftwareName, 106
GetTitle, 105
GetVideoCodecCount, 98
GetVideoCodecName, 98
IsLiveRecording, 101
IsLooping, 103
IsPaused, 102
IsPlaying, 102
IsStopped, 102
OpenAviFile, 97
Pause, 102
Play, 101
ReadImage, 102
SetAudioCodec, 100
SetAudioSource, 99
SetAuthor, 106
SetCallbacks, 94
SetCineFormCodecHDParams, 101
SetCineFormCodecRawParams, 100
SetCopyright, 105
SetCreationDateTime, 105
SetDateTimeOriginal, 106
SetErrorOutput, 94
SetLiveRecording, 101
SetLooping, 103
SetPlaybackFormat, 104
SetPlaybackRate, 104
SetSoftwareName, 106
SetTitle, 105
SetVideoCodec, 98
ShowAudioCodecSettings, 100
ShowVideoCodecSettings, 98
Stop, 102
StopRecording, 97
HAviFileCallbacks, 107
OnAviEndPlayback, 107
OnAviImageRead, 107
HBitmapOverlay, 108
GetBitmapFile, 109
GetLastError, 109
IsBinaryBitmap, 110
SetBitmapFile, 109
SetDisplayColorForBinaryBitmap, 110
SetErrorOutput, 109
SetOverlayPosition, 110
SetTransparentcyColor, 109
SuggestBottomRightPosition, 112
SuggestCenterPosition, 110
WriteBitmapOverlay, 112
HDaqDataSet, 112
CloneDaqDataSet, 114, 117
CreateDaqDataSet, 114, 117
DeleteDaqDataSet, 115, 117
GetChannelNum, 115, 118
GetSample, 115, 118
GetSampleCount, 115, 118
GetTimeStamp, 115, 118
GetTimeStampMS, 116, 118
GetTimeStampUS, 116, 119
IsDummyDataSet, 117, 120
SetDaqDataSet, 116, 119
SetDummyDataSet, 116, 119
SetReferenceFrameIndex, 115, 118
SetTimeStamp, 116, 119
HDaqFile, 120
Close, 123, 130
Create, 123, 129
GetAllocatedDataSets, 125, 131
GetLastError, 123, 129
GetMetadata, 129, 136
GetMetadataLength, 129, 135
GetPlaybackPosition, 127, 134
GetPositionAtTime, 126, 133
GetReadingPosition, 125, 132
GetRecordingPosition, 125, 131
GetRecordingPositionByRefFrame, 125, 132
GetRecordingPositionByRefTimeStamp, 126, 132
GetReferenceInfo, 126, 133
GetSampleRate, 125, 132
GetTimeAtPosition, 127, 133
IsPlaying, 128, 134
Open, 123, 130
Read, 124, 125, 130, 131
RemoveMetadata, 128, 135

ResynchronizePlayback, 128, 135
SetErrorOutput, 123, 129
SetMetadata, 129, 135
SetMetadataSize, 128, 135
SetPlaybackPosition, 127, 134
StartPlayback, 127, 134
StopPlayback, 127, 134
Write, 123, 130
HDaqFileCallbacks, 136
OnDaqPlaybackUpdate, 136, 137
HDeliverCallback, 137
OnPrepareCompress, 137
HEnums.h, 405
eAdvanceSettingsFeatureType, 420
eAudFileMode, 424
eAudioFileFormat, 424
eDAQDeviceParamType, 425
eDeviceAcclType, 426
eFastColorCorrectionMode, 421
eHAudioCompression, 424
eHAudioFormat, 424
eHAudioPlayEventCode, 425
eHAviPlaybackFormat, 421
eHBayerPatternOrigin, 419
eHBayerSampleFactor, 419
eHCayer_Compression, 426
eHCompression, 418
eHDescriptionFormat, 418
eHErrorCode, 427
eHExportFormat, 425
eHGRABBER_NOTIFICATION, 423
eHGrabberInfoType, 416
eHImageDateTimeFormat, 421
eHImageFormat, 417
eHInputEvent, 419
eHInputLevel, 420
eHJpegQuality, 417
eHMerge_Mode, 424
eHMetadataFormat, 426
eHMirror_Mode, 423
eHMotionDetectionMode, 422
eHMovieMode, 421
eHOutputLevel, 420
eHProtectionStatus, 419
eHResizeInterpolationMode, 422
eHRotationInterpolationMode, 422
eHSUGGESTED_ACTION, 423
eHSobelMode, 422
eHSpecialCopyFormat, 420
eHSupportedModule, 413
eJpegDecodeAlgoType, 427
eJpegEncodeAlgoType, 426
eJpegImageExportType, 427
eMonoToPseudoColorLUT, 422
eTimeSourceParamType, 425
H_ABSOFTWARE_ACTIVE_GIGE, 413
H_ABSOFTWARE_ACTIVE_USB, 413
H_ACTIVESILICON_FIREBIRD, 413
H_ACTIVESILICON_PHOENIX, 413
H_ADVANCE_BOOL_VALUE, 421
H_ADVANCE_COMMAND, 421
H_ADVANCE_DOUBLE_VALUE, 421
H_ADVANCE_ENUMERATION, 421
H_ADVANCE_INTEGER_VALUE, 421
H_ADVANCE_OTHERS, 421
H_AJA_KONA, 413
H_ANDOR_LUCA, 413
H_ANDOR_NEO, 413
H_AUD_FILE_AUD_BLOCK, 424
H_AUD_FILE_BWAVE_CONTINOUS, 424
H_AUDIO_COMPRESSION_AAC, 425
H_AUDIO_COMPRESSION_MP3, 425
H_AUDIO_COMPRESSION_NONE, 425
H_AUDIO_FILE_AUD_PCM, 424
H_AUDIO_FILE_BWAVE_PCM, 424
H_AUDIO_FLOAT_MSB_PACKED_PCM, 424
H_AUDIO_FLOAT_MSB_PLANAR_PCM, 424
H_AUDIO_FLOAT_PACKED_PCM, 424
H_AUDIO_FLOAT_PLANAR_PCM, 424
H_AUDIO_INT_MSB_PACKED_PCM, 424
H_AUDIO_INT_MSB_PLANAR_PCM, 424
H_AUDIO_INT_PACKED_PCM, 424
H_AUDIO_INT_PLANAR_PCM, 424
H_AUDIO_READ_FAILED, 425
H_AUDIO_SAMPLE_BUFFER_CHANGED, 425
H_AUDIO_UINT_PACKED_PCM, 424
H_AUDIO_UINT_PLANAR_PCM, 424
H_AUDIO_UNKNOWN, 424
H_AUDIO_UNSUPPORTED_FORMAT, 425
H_AVI_PLAYBACK_BGR, 422
H_AVI_PLAYBACK_BGRX, 422
H_AVI_PLAYBACK_NATIVE, 422
H_AVT_DCAM, 413
H_AVT_DCAM_UNI, 413
H_AVT_UNIVERSAL, 413
H_AXIS, 413
H_BASLER_BCAM, 413
H_BASLER_GIGE, 413
H_BASLER_IEEE1394, 413
H_BASLER_PYLON, 413
H_BASLER_TOF, 413
H_BAUMER2, 413
H_BAUMER_FW, 413
H_BAYER_PATTERNORIGIN_B, 419
H_BAYER_PATTERNORIGIN_GB, 419
H_BAYER_PATTERNORIGIN_GR, 419
H_BAYER_PATTERNORIGIN_R, 419
H_BAYER_SAMPLEFACTOR_1, 419
H_BAYER_SAMPLEFACTOR_2, 419
H_BAYER_SAMPLEFACTOR_4, 419
H_BAYER_SAMPLEFACTOR_8, 419
H_BITFLOW_CL, 413
H_BITFLOW_CXP, 413
H_BLACK_MAGIC, 413
H_BYTE_FORMAT, 419
H_CCAM, 413

H_CCDDIRECT_MICROPIX, 413
H_CMU_IEEE1394, 413
H_COMPRESSION_CAYER, 418
H_COMPRESSION_H264, 418
H_COMPRESSION_H264_LOOP, 418
H_COMPRESSION_H264_RGB, 418
H_COMPRESSION_H264_RGB_LOOP, 418
H_COMPRESSION_HEVC, 418
H_COMPRESSION_HUFFMAN, 418
H_COMPRESSION_HUFFMAN_FAST, 418
H_COMPRESSION_JPEG, 418
H_COMPRESSION_JPEG_LOSSLESS, 418
H_COMPRESSION_LZ, 418
H_COMPRESSION_LZ_FAST, 418
H_COMPRESSION_MPEG4, 418
H_COMPRESSION_NONE, 418
H_COMPRESSION_RLE, 418
H_COMPRESSION_RLE_FAST, 418
H_COMPRESSION_WAVELET, 418
H_COOKE_PCO, 413
H_COOKE_PIXELFLY, 413
H_COOKE_SENSICAM, 413
H_COOKE_SENSICAM_2, 413
H_CORECO_BANDIT2CV, 413
H_CORECO_BANDIT2MV, 413
H_CORECO_BANDIT2RGB, 413
H_CORECO_GIGE, 413
H_CORECO_PC2CAMLINK, 413
H_CORECO_PC2COMP, 413
H_CORECO_PC2VISION, 413
H_CORECO_PCCAMLINK, 413
H_CORECO_VIPERDIG, 413
H_CORECO_X64, 413
H_CPU_DEVICE_ONLY, 426
H_CYR_VANILLA, 426
H_CYR_VANILLA_LOSSY_HI, 426
H_CYR_VANILLA_LOSSY_LOW, 426
H_CYR_VANILLA_LOSSY_MED, 426
H_DAQ_PARAM_ENUM, 425
H_DAQ_PARAM_FLOAT, 425
H_DAQ_PARAM_INT, 425
H_DATAPATH_VISIONRGB, 414
H_DATETIME_FULL, 421
H_DATETIME_LTC, 421
H_DATETIME_STANDARD, 421
H_DATETIME_TIME_T, 421
H_EMERGENT, 414
H_EMS_PHYNXRGB, 414
H_EMS_XTREMERGB, 414
H_EMVACAM, 414
H_EPIPHAN, 414
H_EPIX_D2X, 414
H_EPIX_RAPTOR, 414
H_EPIX_SV5, 414
H_ERROR_AAC_DECODING, 431
H_ERROR_ADD_ACODEC, 428
H_ERROR_ADD_AUDIO_SOURCE, 428
H_ERROR_ADD_AVISPLITTER, 429
H_ERROR_ADD_FEEDER, 428
H_ERROR_ADD_FILEREADER, 429
H_ERROR_ADD_MUX_FILE, 428
H_ERROR_ADD_SAMPLE_GRABBER, 430
H_ERROR_ADD_VCODEC, 428
H_ERROR_AUDIO_CANTCREATEFILE, 430
H_ERROR_AUDIO_CHANNEL_MASK_MATCH, 430
H_ERROR_AUDIO_FORMAT, 430
H_ERROR_AUDIO_FORMAT_CHANGED, 430
H_ERROR_AUDIO_INPUTSUPPORT, 430
H_ERROR_AUDIO_INPUTSUPPORT_LOAD, 430
H_ERROR_AUDIO_INVALID_FORMAT, 430
H_ERROR_AUDIO_INVALID_INDEX, 430
H_ERROR_AUDIO_LOAD, 430
H_ERROR_AUDIO_NOAUD, 430
H_ERROR_AUDIO_OUTPUTSUPPORT, 430
H_ERROR_AUDIO_OUTPUTSUPPORT_LOAD, 430
H_ERROR_AUDIO_READ_FAILED, 430
H_ERROR_AUDIO_WRITE_AUD, 430
H_ERROR_AUIDO_PROCESS_MORE_DATA, 431
H_ERROR_AUIDO_PROCESS_NO_DATA, 431
H_ERROR_AVI_TAG_NOT_FOUND, 431
H_ERROR_BAYER_CONVERSION, 429
H_ERROR_BIND_SOCKET, 430
H_ERROR_BUFFER_INTERNAL, 427
H_ERROR_CANT_ADD_FRAME, 430
H_ERROR_CANTCREATEFILE, 429
H_ERROR_CLIENT_ALREADY_CONNECTED, 430
H_ERROR_CLOSE_SOCKET, 430
H_ERROR_COLOR_BALANCING, 429
H_ERROR_CONNECT_ACODEC_MUX, 428
H_ERROR_CONNECT_ALEVEL_ACODEC, 430
H_ERROR_CONNECT_ALEVEL_MUX, 430
H_ERROR_CONNECT_ASOURCE_ACODEC, 428
H_ERROR_CONNECT_ASOURCE_ALEVEL, 430
H_ERROR_CONNECT_ASOURCE_MUX, 428
H_ERROR_CONNECT_SOURCE_SPLITTER, 428
H_ERROR_CONNECT_SPLITTER_RENDERER, 428
H_ERROR_CONNECT_VCODEC_MUX, 428
H_ERROR_CONNECT_VSOURCE_MUX, 428
H_ERROR_CONNECT_VSOURCE_VCODEC, 428
H_ERROR_CONNECTION_FAILED, 430
H_ERROR_CONVERT_BGR24, 427
H_ERROR_CONVERT_BGRX, 428
H_ERROR_CONVERT_UNSUPPORTED, 428
H_ERROR_CONVERT_UVY411, 428
H_ERROR_CONVERT_UVY422, 428
H_ERROR_CONVERT_UVY444, 428
H_ERROR_CONVERT_YUV422, 428
H_ERROR_CREATE_BUFFER, 427

H_ERROR_CREATE_CAPTURE_BUILDER, 428
H_ERROR_CREATE_CLASS_ENUM, 428
H_ERROR_CREATE_FEEDER, 428
H_ERROR_CREATE_FILE_ERROR, 431
H_ERROR_CREATE_GRAPH_BUILDER, 428
H_ERROR_CREATE_RENDER, 429
H_ERROR_CREATE_SAMPLE_GRABBER, 430
H_ERROR_CREATE_SOCKET, 430
H_ERROR_CREATE_SYSTEM_ENUM, 428
H_ERROR_CREATEC_UNSUPPORT_FORMAT, 431
H_ERROR_DAQSUPPORT_CANTCREATEFILE, 429
H_ERROR_DAQSUPPORT_FORMAT_CHANGED, 430
H_ERROR_DAQSUPPORT_INVALID_INDEX, 430
H_ERROR_DAQSUPPORT_LOAD, 429
H_ERROR_DAQSUPPORT_NODAQ, 430
H_ERROR_DAQSUPPORT_READ_FAILED, 429
H_ERROR_DAQSUPPORT_WRITE_DAQ, 430
H_ERROR_DECODE_MORE_DATA, 431
H_ERROR_DONTSUPPORTREWRITE, 429
H_ERROR_EXPORT_FAILED, 429
H_ERROR_EXPORT_IN_PROGRESS, 427
H_ERROR_FAILED_TO_ADD_TAG, 431
H_ERROR_FAILEDREGISTER, 427
H_ERROR_FILENOTFOUND, 429
H_ERROR_FRAME_SKIPPED, 431
H_ERROR_GDIDRAW, 429
H_ERROR_GPU_IMAGE_PUSH, 431
H_ERROR_H264_DECODING, 430
H_ERROR_H264_ENCODING, 430
H_ERROR_H264_UNSUPPOT_IMAGE_SIZE, 430
H_ERROR_HEVC_DECODING, 431
H_ERROR_HEVC_ENCODING, 431
H_ERROR_IMAGE_FORMAT, 429
H_ERROR_IMAGEFORMATCHANGED, 429
H_ERROR_INI_FAIL, 427
H_ERROR_INPUTSUPPORT_DEV_INIT_FAILED, 429
H_ERROR_INPUTSUPPORT_DEV_UNINIT_FAILED, 429
H_ERROR_INPUTSUPPORT_LOAD, 429
H_ERROR_INPUTSUPPORT_MONITOR, 429
H_ERROR_INTERLEAVE_MODE, 428
H_ERROR_INTERLEAVE_TIME, 428
H_ERROR_INVALID_COMPRESSION, 427
H_ERROR_INVALID_DIGIT, 427
H_ERROR_INVALID_INDEX, 429
H_ERROR_INVALID_LUT_FILE, 431
H_ERROR_INVALID_SOCKET, 430
H_ERROR_LISTEN_SOCKET, 430
H_ERROR_LOADING_AUDIO_CODEC, 428
H_ERROR_LOADING_AUDIO_SOURCE, 428
H_ERROR_LOADING_VIDEO_CODEC, 428
H_ERROR_MALFORMED_XML_FILE, 431
H_ERROR_MERGE_ERROR, 429
H_ERROR_MIRROR_ERROR, 429
H_ERROR_MONIKER_NULL, 428
H_ERROR_MP3_DECODING, 431
H_ERROR_MP4_AUDIO_READ, 431
H_ERROR_MP4_VIDEO_READ, 431
H_ERROR_MPEG4_DECODING, 431
H_ERROR_MPEG4_ENCODING, 431
H_ERROR_NO_AUDIO_SOURCE, 430
H_ERROR_NO_BEGINEXPORT, 427
H_ERROR_NO_CAPTURE_GRAPH, 431
H_ERROR_NO_CODEC_SETTINGS, 428
H_ERROR_NO_FILE, 430
H_ERROR_NO_GRABBER, 427
H_ERROR_NO_GRABMAN, 427
H_ERROR_NO_IMAGE_DATA, 427
H_ERROR_NO_PROCESSING_APPLIED, 429
H_ERROR_NO_RECORDING, 430
H_ERROR_NOAUTHORIZATION, 427
H_ERROR_NOBITMAP, 429
H_ERROR_NOCOMPORT, 429
H_ERROR_NOSEQ, 428
H_ERROR_NOT_CONNECTED, 431
H_ERROR_OUT_OF_RANGE, 430
H_ERROR_OUTOFMEMORY, 429
H_ERROR_OUTPUTSUPPORT_DEV_INIT_FAILED, 429
H_ERROR_OUTPUTSUPPORT_DEV_UNINIT_FAILED, 429
H_ERROR_OUTPUTSUPPORT_LOAD, 429
H_ERROR_QUICKTIME_ERROR, 429
H_ERROR_QUICKTIME_INIFAIL, 429
H_ERROR_REGISTRY_READ, 431
H_ERROR_REGISTRY_WRITE, 431
H_ERROR_RESIZE_ERROR, 429
H_ERROR_ROTATION_ERROR, 429
H_ERROR_SAMPLE_GRABBER_CALLBACK, 430
H_ERROR_SCRIPTER_NOT_RUNNING, 429
H_ERROR_SCRIPTRUNNING, 429
H_ERROR_SECTORBYTES_ZERO, 429
H_ERROR_SEND, 430
H_ERROR_SEQ_CANTCREATEFILE, 428
H_ERROR_SEQ_INVALIDDATA, 428
H_ERROR_SEQ_INVALIDDEPTH, 428
H_ERROR_SEQ_INVALIDFRAMERATE, 428
H_ERROR_SEQ_INVALIDHEIGHT, 428
H_ERROR_SEQ_INVALIDIMAGESIZE, 428
H_ERROR_SEQ_INVALIDLENGTH, 428
H_ERROR_SEQ_INVALIDMAGICNUMBER, 428
H_ERROR_SEQ_INVALIDWIDTH, 428
H_ERROR_SEQ_UNSUPPORTED_FORMAT, 428
H_ERROR_SEQ_WRONGHEADERSIZE, 428
H_ERROR_SERVER_ALREADY_OPEN, 430
H_ERROR_SERVER_NOT_RUNNING, 430
H_ERROR_SHUTDOWN_SOCKET, 430
H_ERROR_SOCKET_ALREADY_CREATED, 431

H_ERROR_STOP_PLAYBACK, 429
H_ERROR_STREAMING, 429
H_ERROR_SUCCESS, 427
H_ERROR_SYNCH_SOURCE, 428
H_ERROR_TOOMUCHDATA, 429
H_ERROR_UNSUPPORT_FUNCTION, 430
H_ERROR_UNSUPPORTED, 429
H_ERROR_UNSUPPORTED_ADJUSTMENTS, 429
H_ERROR_UNSUPPORTED_ADVANCEFEATURE, 429
H_ERROR_UNSUPPORTED_INTERFACE, 431
H_ERROR_UNSUPPORTED_SETTINGS, 429
H_ERROR_VIDEOOUTPUTSUPPORT_LOAD, 429
H_ERROR_WRITE_ERROR, 431
H_ERROR_WRITE_SEQ, 428
H_ERROR_WRONG_MODE, 429
H_EURESYS_GRABLINKVALUE, 414
H_EURESYS_IOTA, 414
H_EURESYS_PICOLO, 414
H_EURESYS_PICOLO_HD, 414
H_EXPORT_AVI, 425
H_EXPORT_BMP, 425
H_EXPORT_DPX, 425
H_EXPORT_FIT, 425
H_EXPORT_JP2, 425
H_EXPORT_JPG, 425
H_EXPORT_MOV, 425
H_EXPORT_PNG, 425
H_EXPORT_SEQ, 425
H_EXPORT_TIFF, 425
H_FAST_CORRECTION_AUTO, 421
H_FAST_CORRECTION_MANUAL, 421
H_FAST_CORRECTION_NONE, 421
H_FCAV_ARINC818, 414
H_FOCULUS_CAMERA, 414
H_FORESIGHTIMAGING_ACCUSTREAM, 414
H_FR_STREAM_USB, 414
H_GEVICAM, 414
H_GIGALINX, 414
H_GIGALINX_SONY, 414
H_GPU_DEVICE_AMD, 426
H_GPU_DEVICE_INTEL, 426
H_GPU_DEVICE_NVIDIA_CUDA, 426
H_GPU_DEVICE_NVIDIA_OPENCL, 426
H_GPU_DEVICE_UNKNOWN, 426
H_GRABBER_INFO_TYPE_DOUBLE, 417
H_GRABBER_INFO_TYPE_LONG, 417
H_GRABBER_INFO_TYPE_STRING, 417
H_GRABBER_NOTIFICATION_CAPTURE_ERROR, 423
H_GRABBER_NOTIFICATION_CAPTURE_EVENT_RECEIVED, 423
H_GRABBER_NOTIFICATION_CONNECTED, 423
H_GRABBER_NOTIFICATION_CONNECTION_LOST, 423
H_GRABBER_NOTIFICATION_DISCONNECTED, 423
H_GRABBER_NOTIFICATION_FRAME_DROP, 423
H_GRABBER_NOTIFICATION_INVALID_SIGNAL, 423
H_GRABBER_NOTIFICATION_PTP_CLOCK_STATUS, 423
H_GRABBER_NOTIFICATION_SIGNAL_PRESENT, 423
H_GRABBER_NOTIFICATION_TIMESTAP_SET, 423
H_GRT_MATRIX, 414
H_GRT_MATRIX_HL2V, 414
H_HAMAMATSU_DCAM, 414
H_HASOTEC_FG30, 414
H_HITACHI_DCAM, 414
H_HL2V, 414
H_IDS_UEYE, 414
H_IMAGE_BGR, 417
H_IMAGE_BGR10_PPACKED, 418
H_IMAGE_BGR10_PPACKED_PHOENIX, 418
H_IMAGE_BGR555_PACKED, 417
H_IMAGE_BGR565_PACKED, 417
H_IMAGE_BGRx, 417
H_IMAGE_GRT_DUAL_IMAGE_RGB, 418
H_IMAGE_GVSP_BGR10V1_PACKED, 418
H_IMAGE_GVSP_BGR10V2_PACKED, 418
H_IMAGE_GVSP_PIX_MONO10_BAYER_PACKED, 418
H_IMAGE_GVSP_PIX_MONO10_PACKED, 418
H_IMAGE_MONO, 417
H_IMAGE_MONO80_64LSB_16MSB, 418
H_IMAGE_MONO_BAYER, 417
H_IMAGE_MONO_BAYER_MSB, 417
H_IMAGE_MONO_BAYER_MSB_SWAP, 417
H_IMAGE_MONO_BAYER_PPACKED, 418
H_IMAGE_MONO_BAYER_PPACKED_8448, 418
H_IMAGE_MONO_BAYER_PPACKED_8844, 418
H_IMAGE_MONO_MSB, 417
H_IMAGE_MONO_MSB_SWAP, 417
H_IMAGE_MONO_PPACKED, 418
H_IMAGE_MONO_PPACKED_8448, 418
H_IMAGE_MONO_PPACKED_8844, 418
H_IMAGE_PHOTON_FOCUS_DR_BAYER, 418
H_IMAGE_PHOTON_FOCUS_DR_MONO, 418
H_IMAGE_PHOTON_FOCUS_QR_BAYER, 418
H_IMAGE_PHOTON_FOCUS_QR_MONO, 418
H_IMAGE_PLANAR, 417
H_IMAGE_RGB, 417
H_IMAGE_RGB10_PPACKED_PHOENIX, 418
H_IMAGE_UNKNOWN, 417
H_IMAGE_UVY411, 417
H_IMAGE_UVY422, 417
H_IMAGE_UVY444, 417
H_IMAGE_YUV420_PLANAR, 417
H_IMAGE_YUV422, 417
H_IMAGINGSOURCE, 414

H_IMITECH_1394, 414
H_IMITECH_GIGE, 414
H_IMITECH_NEPTUNE, 414
H_IMPERX_BOBCAT_GIGE, 414
H_IMPERX_FRAMELINK, 414
H_IMPERX_FRAMELINK_EXPRESS, 414
H_IMPERX_GIGE, 414
H_IMPERX_SDI_EXPRESS, 414
H_IMPERX_VCE_EXPRESS, 414
H_IMPERX_VCEPRO, 414
H_INPUT_EVENT_FALLING_EDGE, 420
H_INPUT_EVENT_RISING_EDGE, 420
H_INPUT_LEVEL_HIGH, 420
H_INPUT_LEVEL_LOW, 420
H_INPUT_LEVEL_UNDEFINED, 420
H_INTEGRAL_FLASHBUSDX, 414
H_INTEGRAL_FLASHBUSMV, 414
H_INTEGRAL_FLASHBUSMX, 414
H_INTEGRAL_SPECTRIM, 414
H_IP_CAMERA, 414
H_ISG_1394, 415
H_JAIGENICAM2, 415
H_JAIGENICAM_GIGE, 415
H_JAIPULNIX_GIGE, 415
H_JENOPTIK_DIJ, 415
H_JENOPTIK_PROGRESC10, 415
H_JENOPTIK_PROGRESC14, 415
H_JENOPTIK_PROGRESCXX, 415
H_JPEG_ACCURATE, 417
H_JPEG_DE_ALGO_GPU_INTEL, 427
H_JPEG_DE_ALGO_IPP, 427
H_JPEG_DE_ALGO_TURBO, 427
H_JPEG_DEFAULT, 417
H_JPEG_EN_ALGO_GPU, 427
H_JPEG_EN_ALGO_IPP, 427
H_JPEG_EN_ALGO_TURBO, 427
H_JPEG_FAST, 417
H_JPEG_IMAGE_EXPORT_FREE_IMAGE, 427
H_JPEG_IMAGE_EXPORT_GPU, 427
H_JPEG_IMAGE_EXPORT_IPP, 427
H_JPEG_IMAGE_EXPORT_TURBO, 427
H_JPEG_QUALITYAVERAGE, 417
H_JPEG_QUALITYBAD, 417
H_JPEG_QUALITYGOOD, 417
H_JPEG_QUALITYNORMAL, 417
H_JPEG_QUALITYSUPERB, 417
H_KAPPA_FW, 415
H_KAPPA_HIRES3XR, 415
H_KAYA, 415
H_KINECT, 415
H_LEUTRON_PICSIGHT_GIGE, 415
H_LEUTRON_SIMPLON_GIGE, 415
H_LUMENERA_GIGE, 415
H_LUMENERA_USB2, 415
H_MATRIXVISION, 415
H_MATRIXVISION_BLUECOUGAR, 415
H_MATRIXVISION_BLUEFOX, 415
H_MATRIXVISION_HYPERION, 415
H_MATRIXVISION_SIGMA, 415
H_MATRIXVISION_TITAN, 415
H_MATROX_CL, 415
H_MATROX_CXP, 415
H_MATROX_HELIOS, 415
H_MATROX_METEOR2_CL, 415
H_MATROX_METEOR2_DIG, 415
H_MATROX_METEOR2_MC, 415
H_MATROX_METEOR2_STD, 415
H_MATROX_MORPHIS, 415
H_MATROX_ODYSSEY, 415
H_MATROX_ORION, 415
H_MATROX_RADIENT, 415
H_MATROX_SOLIOS_XCL, 415
H_MATROX_VIO_XCL, 415
H_MBCS_FORMAT, 419
H_MERGE_HORZ, 424
H_MERGE_VERT, 424
H_METADATA_BINARY, 426
H_METADATA_BOOL, 426
H_METADATA_BYTE, 426
H_METADATA_DOUBLE, 426
H_METADATA_INT, 426
H_METADATA_INT64, 426
H_METADATA_SHORT, 426
H_METADATA_STRING, 426
H_METADATA_UINT, 426
H_METADATA_UINT64, 426
H_METADATA_UNKNOWN, 426
H_METADATA_USHORT, 426
H_MIGHTEX, 415
H_MIGHTEX_CCD, 415
H_MIRROR_BOTH, 424
H_MIRROR_HORZ, 424
H_MIRROR_VERT, 424
H_MONO_TO_PSEUDO_COLOR_DISABLED, 423
H_MONO_TO_PSEUDO_COLOR_LUT_COLD, 423
H_MONO_TO_PSEUDO_COLOR_LUT_HOT, 423
H_MONO_TO_PSEUDO_COLOR_LUT_INVER↩TED_RAINBOW, 423
H_MONO_TO_PSEUDO_COLOR_LUT_NEGA↩TIVE, 423
H_MONO_TO_PSEUDO_COLOR_LUT_RAINB↩OW, 423
H_MONO_TO_PSEUDO_COLOR_LUT_SATU↩RATION_3, 423
H_MONO_TO_PSEUDO_COLOR_LUT_SATU↩RATION_5, 423
H_MOTION_DETECTION_ABS_DIFF, 422
H_MOTION_DETECTION_EDGE, 422
H_MOTION_DETECTION_EDGE_ADAPTIVE, 422
H_MOTION_DETECTION_MAX_ABS_DIFF, 422
H_MOTION_DETECTION_MEAN, 422
H_MOVIE_PLAYBACK, 421
H_MOVIE_RECORDING, 421

H_MOVIE_UNDEFINED, 421
H_NET_GIGE, 415
H_NET_ICUBE, 415
H_NI_GRABBERS_0, 415
H_NIKON_D7000, 415
H_NORPIX_VIRTUAL, 415
H_OLYMPUS_DP70, 415
H_ONVIF, 415
H_OPTRONICS_MACROFIRE, 415
H_OPTRONICS_MAGNAFIRE, 416
H_OPTRONICS_MICROFIRE, 416
H_OUTPUT_LEVEL_HIGH, 420
H_OUTPUT_LEVEL_LOW, 420
H_OUTPUT_LEVEL_UNDEFINED, 420
H_PICOLO_H264_UX, 414
H_PICOLO_H264_V16, 414
H_PIXELINK_PLA500, 416
H_PIXELINK_PLA600, 416
H_PIXELINK_PLA700, 416
H_PIXELINK_TM1020, 416
H_PIXELINK_VCA1281, 416
H_PIXERA_PENGUIN, 416
H_PLEORA_EBUS, 416
H_PLEORA_PT1000CL, 416
H_PRINCETON, 416
H_PROSILICA, 416
H_PROSILICA_GIGE, 416
H_PROTECTION_INVALID, 419
H_PROTECTION_NONE, 419
H_PROTECTION_VALID, 419
H_PTGREY, 416
H_PTGREY_LADYBUG, 416
H_QIMAGING_MP, 416
H_QIMAGING_RETIGA, 416
H_REDLAKE_MEGAPLUSII, 416
H_RESIZE_INTER_CUBIC, 422
H_RESIZE_INTER_LANCZOS, 422
H_RESIZE_INTER_LINEAR, 422
H_RESIZE_INTER_NN, 422
H_RESIZE_INTER_SUPER, 422
H_ROPER_CASCADE, 416
H_ROPER_COOLSNAPCF, 416
H_ROTATION_INTER_CUBIC, 422
H_ROTATION_INTER_LINEAR, 422
H_ROTATION_INTER_NN, 422
H_SBF_FIREWIRE, 416
H_SCION_CFW, 416
H_SILICONSOFTWARE_MICROENABLE, 416
H_SILICONSOFTWARE_MICROENABLE5, 416
H_SILLICONIMAGING_GIGE, 416
H_SOBEL_BOTH, 422
H_SOBEL_HORIZONTAL, 422
H_SOBEL_VERTICAL, 422
H_SPECIAL_COPY_BGR555, 420
H_SPECIAL_COPY_BGR565, 420
H_SPECIAL_COPY_BGRA, 420
H_SPECIAL_COPY_BYR1, 420
H_SPECIAL_COPY_BYR2, 420
H_SPECIAL_COPY_BYR4, 420
H_SPECIAL_COPY_H264, 420
H_SPECIAL_COPY_JPEG, 420
H_SPECIAL_COPY_RG30, 420
H_SPECIAL_COPY_RG48, 420
H_SPECIAL_COPY_UVY422, 420
H_SPECIAL_COPY_Y16, 420
H_SPECIAL_COPY_Y8, 420
H_SPECIAL_COPY_YUV411, 420
H_SPECIAL_COPY_YUV422, 420
H_SPECIAL_COPY_YUV444, 420
H_SPINNAKER, 416
H_SPOT, 416
H_SUGGESTED_ACTION_DISABLE_FRAME_↩ DROP, 423
H_SUGGESTED_ACTION_ENABLE_FRAME_↩ DROP, 423
H_SUGGESTED_ACTION_NONE, 423
H_SUGGESTED_ACTION_OPEN_HARDWAR↩ ESETTINGDLG, 423
H_SUGGESTED_ACTION_PAUSE_STREAM, 423
H_SUGGESTED_ACTION_RELOAD_GRABBER, 423
H_SUGGESTED_ACTION_RESTART_STREAM, 423
H_SUGGESTED_ACTION_RESUME_STREAM, 423
H_SUGGESTED_ACTION_UNLOAD_GRABBER, 423
H_SUMIX_12A2, 416
H_SUMIX_150, 416
H_SUMIX_15M5X, 416
H_SUMIX_16EXX, 416
H_SUMIX_M7X, 416
H_SUMIX_M8X, 416
H_SVS_VISTEK, 416
H_SVS_VISTEK_GIGE, 416
H_SYNC_STATUS_FREERUN, 419
H_SYNC_STATUS_HOLDOVER, 419
H_SYNC_STATUS_SYNCHRONIZED, 419
H_SYNC_STATUS_UNKOWN, 419
H_SYNC_STATUS_UNSUPPORTED, 419
H_TATTILE_DUPLEX, 416
H_TS_PARAM_ENUM, 426
H_TS_PARAM_FLOAT, 426
H_TS_PARAM_INT, 426
H_UNICODE_FORMAT, 419
H_VIEWORKS, 416
H_VIMBA, 416
H_VRMAGIC_RIC10, 416
H_WDM_CAMERA, 416
H_XCELERA, 416
H_XENICS, 416
H_XIMEA, 416
HTimeSourceSyncStatus, 419
HErrorManager, 137
GetErrorString, 138

HFPSMonitor, 138
EndBench, 140
GetBenchTime, 140
GetLastError, 139
NewLiveFrame, 139
SetCallbacks, 141
SetErrorOutput, 139
StartBench, 140
StartLiveCheck, 139
HFPSMonitorCallbacks, 141
OnNewFrameRate, 141
HFrameAverage, 142
AllocateAverageBuffer, 143, 144
FreeAverageBuffer, 143, 144
GetLastError, 143, 144
RollingFrameAverage, 144, 145
SetErrorOutput, 143, 144
StandardFrameAverage, 143, 145
HGPUDeviceManager, 145
DisableAMDOpenCLEncoding, 149
DisableNvidiaCudaEncoding, 148
DisableNvidiaOpenCLEncoding, 147
EnableAMDOpenCLEncoding, 148
EnableNvidiaCudaEncoding, 148
EnableNvidiaOpenCLEncoding, 147
GetAMDGPUDeviceInfo, 148
GetIntelGPUDeviceInfo, 147
GetLastError, 147
GetNvidiaGPUDeviceInfo, 147
IsAMDOpenCLEncodingEnabled, 148
IsNvidiaCudaEncodingEnabled, 148
IsNvidiaOpenCLEncodingEnabled, 147
SetErrorOutput, 147
HGrabCallbacks, 149
OnAfterQueuingGrab, 151
OnAfterStartStreaming, 151
OnAfterStopStreaming, 152
OnBeforeImageReceived, 150
OnBeforeQueuingGrab, 151
OnBeforeStartStreaming, 151
OnBeforeStopStreaming, 151
OnGPSSentencesReceived, 152
OnImageReceived, 150
OnImageReleased, 150
OnNotification, 152
HGrabModule, 152
Accessible, 176
CloseAdvanceSettingsWindow, 175
CreateAdvanceSettingsWindow, 174
DoModalAdvanceSettingsWindow, 174
EndQueryMode, 181
ExecuteCommandFeature, 180
ExportGrabberSettings, 160
GetAdjustmentCaps, 167
GetAdjustmentCount, 167
GetAutomaticAdjustment, 169
GetBinning, 165
GetBoolFeatureValue, 179
GetBufferCount, 182
GetBufferUsage, 182
GetCategoriesCap, 175
GetCategoriesCount, 175
GetCommunicationType, 173
GetCurrentGrabber, 160
GetCurrentValues, 166
GetDoubleAdjustment, 168
GetDoubleFeatureCap, 179
GetDoubleFeatureValue, 179
GetDriverCount, 158
GetDriverInfo, 159
GetEndChar, 172
GetEnumFeatureValue, 177
GetEnumFeatureValueCap, 177
GetEnumFeatureValueCount, 177
GetFeatureCap, 176
GetFeatureCount, 176
GetGPSData, 171
GetGrabberCount, 181
GetGrabberInfoCount, 162
GetGrabberInfoName, 162
GetGrabberInfoValueDouble, 163
GetGrabberInfoValueLong, 163
GetGrabberInfoValueString, 163
GetGrabberInfoValueType, 163
GetGrabberName, 181
GetIODriver, 180
GetImageBitDepth, 170
GetImageBitDepthReal, 170
GetImageFormat, 170
GetImageHeight, 170
GetImageSizeBytes, 170
GetImageWidth, 170
GetIntegerAdjustment, 168
GetIntegerFeatureCap, 178
GetIntegerFeatureValue, 178
GetLastError, 158
GetMaxBufferUsage, 182
GetMaxResolution, 165
GetPanTilt, 184
GetPanTiltXRange, 183
GetProgFolder, 173
GetROI, 164
GetSavedProgFolder, 173
GetSettingsCaps, 165
GetSettingsCount, 165
GetTextMode, 172
GetTotalBufferCount, 182
GetValuesCaps, 166
GetValuesCount, 166
GetZoom, 184
GetZoomRange, 183
IsCOMPortAvailable, 171
IsGPSSupported, 171
IsStreaming, 161
IsSupportAdvanceSettings, 174
IsSupportPanTilt, 182

IsSupportZoom, 183
Load, 159
OneShotAdjustment, 169
ReSizeAdvanceSettingsWindow, 174
Receive, 172
SetAutomaticAdjustment, 169
SetBinning, 164
SetBoolFeatureValue, 180
SetBufferCount, 182
SetCallbacks, 161
SetCurrentValues, 166
SetDefaultGrabber, 181
SetDoubleAdjustment, 168
SetDoubleFeatureValue, 179
SetEnumFeatureValue, 177
SetErrorOutput, 158
SetIntegerAdjustment, 168
SetIntegerFeatureValue, 178
SetPanTilt, 183
SetROI, 164
SetZoom, 184
ShowAdvanceSettingsWindow, 175
ShowProperties, 162
StartQueryMode, 180
StartStreaming, 161
StopReadingGPSData, 171
StopStreaming, 161
Transmit, 171
Unload, 160
HGrayConverter, 184
CheckGrayConvertionImageFormat, 185, 186
GetLastError, 185, 186
GrayToMono8, 185, 186
SetErrorOutput, 185, 186
HIODriver, 242
DaqPresent, 251
GetCurrentLevel, 247
GetDaqDeviceCount, 251
GetDaqDeviceEnumParameterItemCount, 255
GetDaqDeviceEnumParameterItemInfo, 255
GetDaqDeviceEnumParameterValue, 256
GetDaqDeviceFloatParameterItemInfo, 255
GetDaqDeviceFloatParameterValue, 256
GetDaqDeviceIntParameterItemInfo, 256
GetDaqDeviceIntParameterValue, 257
GetDaqDeviceLineCount, 252
GetDaqDeviceLineInfo, 252
GetDaqDeviceLineRangeSettingsCount, 252
GetDaqDeviceLineRangeSettingsInfo, 253
GetDaqDeviceName, 251
GetDaqDeviceParametersCount, 254
GetDaqDeviceParametersInfo, 254
GetDeviceInputLineCount, 246
GetDeviceInputLineInfo, 247
GetDeviceOutputLineCount, 249
GetDeviceOutputLineInfo, 249
GetDriverName, 245
GetInputDeviceCount, 245
GetInputDeviceName, 246
GetMaxSampleRate, 253
GetOutputDeviceCount, 248
GetOutputDeviceName, 248
GetOutputLevel, 249
GetPollingDelay, 247
GetSample, 254
HIODriver, 245
InitializeDaqDevice, 251
InitializeDeviceInput, 246
InitializeDeviceOutput, 248
InitializePulseGenerator, 250
InputDevicePresent, 245
Monitor, 247
OutputDevicePresent, 248
PulseGeneratorPresent, 250
SetDaqDeviceEnumParameterValue, 256
SetDaqDeviceFloatParameterValue, 257
SetDaqDeviceIntParameterValue, 257
SetOutputLevel, 250
SetTimeSourceDLL, 252
StartCapture, 253
StartPulseGenerator, 250
StopCapture, 254
StopPulseGenerator, 251
UnInitializeDaqDevice, 252
UnInitializeDeviceInput, 246
UnInitializeDeviceOutput, 249
UnInitializePulseGenerator, 250
HImage, 186
AddMetadata, 193
CloneHImage, 189
CreateHImage, 188
DeleteHImage, 189
GetBlueChannel, 191
GetGreenChannel, 190
GetImageBitDepth, 189
GetImageBitDepthReal, 190
GetImageFormat, 190
GetImageHeight, 189
GetImageSizeBytes, 190
GetImageWidth, 189
GetMetadata, 192
GetMetadataCount, 192
GetRawImageData, 190
GetRedChannel, 190
GetSpecificMetadata, 193
GetSpecificMetadataCount, 193
GetTimestamp, 191
GetTimestampMS, 192
GetTimestampUS, 192
HasImageDecoded, 191
ReallocateCompressedImage, 191
ReallocateImage, 191
SetTimestamp, 192
HImageColorProcessing, 193
AddThirdPartLutFile, 206
ApplyColorProcessing, 197

Enable3DLookupTable, 204
EnableAutoColorBalance, 202
EnableBGRxConversion, 199
EnableBayerConversion, 197
EnableColorProcessingHQConversion, 200
EnableLUTProcess, 204
EnableMatrixColorCorrection, 200
EnableMonochromeConversion, 209
ExportLUTToTextFile, 208
Get1DLut, 209
Get1DPreDefinedLUT, 207
Get1DPreDefinedLUTCount, 207
Get1DSupportThirdPartLutFileCount, 205
Get1DSupportThirdPartLutFileFormatInfo, 205
Get3DSupportThirdPartLutFileCount, 205
Get3DSupportThirdPartLutFileFormatInfo, 205
GetASCSettingsParam, 203
GetAutoColorBalanceAlgorithm, 202
GetAutoWhiteBalanceROI, 203
GetCalcMode, 199
GetColorClipLevel, 207
GetCorrectionMatrix, 201
GetLastError, 196
GetManualColorBalance, 202
GetMonoPseudoColorMode, 207
GetPatternOrigin, 199
GetSampleFactor, 199
Is3DLookupTable, 204
IsAutoColorBalanceEnabled, 202
IsBGRxConversionEnabled, 200
IsBayerConversionEnabled, 197
IsColorProcessingHQConversionEnabled, 200
IsLUTProcessEnabled, 204
IsMatrixColorCorrectionEnabled, 200
IsMonochromeConversionEnabled, 209
Load1DPreDefinedLUT, 208
LoadNpxTextLutFile, 208
RemoveThirdPartLutFile, 206
SetASCSettingsParam, 204
SetAutoColorBalanceAlgorithm, 203
SetAutoWhiteBalanceROI, 203
SetCacheLUTFileName, 205
SetCalcMode, 199
SetColorClipLevel, 206
SetCorrectionMatrix, 201
SetErrorOutput, 196
SetManualColorBalance, 202
SetMonoPseudoColorMode, 207
SetPatternOrigin, 197
SetSampleFactor, 199
HImageCompressionString
Utilities, 19
HImageConverter, 209
ConvertToBGR, 210
ConvertToBGRx, 211
ConvertToMono8, 211
GetLastError, 210
SetErrorOutput, 210
HImageExporter, 211
AddJPEGMetadata, 222
BeginExportBmp, 213
BeginExportBmpTo, 217
BeginExportDng, 216
BeginExportDngTo, 219
BeginExportDpx, 216
BeginExportDpxTo, 219
BeginExportFit, 215
BeginExportFitTo, 218
BeginExportJpeg, 214
BeginExportJpeg2k, 215
BeginExportJpeg2kTo, 218
BeginExportJpegTo, 218
BeginExportJpegXR, 216
BeginExportJpegXRTo, 219
BeginExportPng, 215
BeginExportPngTo, 218
BeginExportTiff, 214
BeginExportTiffTo, 218
EnableGPUJpegCompressionBayerInterp, 221
EnableGPUJpegCompressionColorprcessingLUT, 222
EnableTIFFMetadata, 222
EndExport, 217
EndExportTo, 220
ExportImage, 217
ExportImageTo, 220
GetDateTimeFormat, 222
GetJPEGQuality, 221
GetLastError, 213
SetDateTimeFormat, 223
SetErrorOutput, 213
SetJPEGQuality, 220
SetJpegEncoderType, 221
SetJpegGPUCompressionBayerPatternType, 221
SetJpegGPUEncoderType, 221
SetSeparator, 223
UpdateGPUJpegCompressionLUT, 222
HImageFormatString
Utilities, 19
HImageMerge, 223
CheckImageMergeFormat, 224
ForceOutputBRG24, 224
GetLastError, 224
GetMergeMode, 224
Merge, 225
MultipleMerge, 225
SetErrorOutput, 224
SetMergeMode, 224
Stitch, 225
HImageMirror, 226
CheckImageMirrorFormat, 227
GetLastError, 226
GetMirrorMode, 226
Mirror, 227
SetErrorOutput, 226
SetMirrorMode, 226

HImageOverlay, 227
BeginDrawing, 233
EndDrawing, 233
GetAlpha, 230
GetCrosshairThickness, 232
GetLastError, 229
GetLineColor, 233
GetLineThickness, 232
GetOverlayColor, 230
GetPolygonPixelCount, 233
GetTextBackgroundColor, 230
SetAlpha, 230
SetCrosshairThickness, 231
SetErrorOutput, 229
SetLineColor, 232
SetLineStyle, 232
SetLineThickness, 232
SetOverlayColor, 229
SetTextBackgroundColor, 230
SetTextOverlayAlign, 230
SetTextOverlayFont, 231
SetTextOverlaySize, 231
WriteAlignedMultilinesTextOverlay, 234
WriteAlignedTextOverlay, 233
WriteAngleOverlay, 237
WriteArrowOverlay, 236
WriteBezierOverlay, 238
WriteCircleOverlay, 237
WriteCrosshair, 235
WriteCustomOverlay, 239
WriteEllipseOverlay, 236
WriteFilledEllipseOverlay, 237
WriteFilledPolyOverlay, 238
WriteFilledRectOverlay, 236
WriteFlatEndsLineOverlay, 235
WriteLineOverlay, 235
WriteMultilineTextOverlay, 234
WritePolyOverlay, 238
WriteRectOverlay, 236
WriteRoundEndsLineOverlay, 235
WriteTextOverlay, 234
HImageResize, 239
CheckImageReSampleFormat, 242
CheckImageResizeFormat, 241
GetDestinationSize, 241
GetInterpolationMode, 241
GetLastError, 240
GetSourceRoi, 240
IsKeepImageRatio, 241
ReSample, 242
Resize, 242
SetDestinationSize, 240
SetErrorOutput, 240
SetInterpolationMode, 241
SetKeepImageRatio, 241
SetSourceRoi, 240
HItem, 258
GetDescription, 258
GetID, 258
HJpegCompression, 258
Decode, 260
Encode, 259
GetLastError, 259
IsFormatSupported, 259
SetErrorOutput, 259
HLineScanManager, 260
Configure, 262, 265
ConfigureAVIExport, 264, 267
ConfigureImageExport, 264, 267
ConfigureMOVExport, 264, 267
Export, 265, 268
GetCurrentExportedFrame, 265, 268
GetImageFromSeq, 263, 266
GetLiveImage, 263, 266
GetTimeStampImages, 265, 268
PushImageToQueue, 263, 266
SetDirection, 263, 266
SetLineCount, 263, 266
HLocker, 268
TryToLock, 269
HMP3File, 324
AddData, 326
Close, 328
Create, 325
EndRecording, 326
GetAudioInfo, 327
GetLastError, 325
GetPositionByTimeStamp, 328
GetReadingPosition, 328
GetTimeStamp, 327
GetTotalAudioDatas, 327
Open, 326
ReadAudio, 327
SetAudioParam, 326
SetErrorOutput, 325
SetReadingPosition, 328
HMP4File, 328
AddAudio, 332
AddFrame, 332
Close, 333
Create, 330
EndRecording, 333
FinalizeWrite, 333
GetActionMode, 335
GetAllocatedFrames, 336
GetAudioInfo, 334
GetIndexAt, 335
GetKeyFrameByIndex, 336
GetLastError, 330
GetTotalAudioDatas, 336
GetVideoInfo, 334
IsPlaying, 337
IsSuggestPushAudio, 332
IsSuggestPushVideo, 332
Open, 333
Play, 336

PlayFixedRate, 336
PrepareWrite, 332
Read, 334, 335
ReadAudio, 337
SetAudioParam, 331
SetCallbacks, 333
SetErrorOutput, 330
SetPlaybackFrameRate, 336
SetPlaybackPosition, 335
SetVideoParam, 331
Stop, 336
HMPEGTsFile, 337
AddAudio, 340
AddFrame, 339
Close, 340
Create, 338
EndRecording, 340
FinalizeWrite, 340
GetLastError, 338
PrepareWrite, 339
SetAudioParam, 339
SetErrorOutput, 338
SetVideoParam, 339
HMetadata, 269
BuildFrom, 271
GetData, 272
GetDataSizeBytes, 271
GetMetadataString, 272
GetPacketSize, 272
GetUID, 271
HMetadata, 270, 271
operator=, 271
SetMetadataString, 272
WritePacketTo, 272
HMetadataGenericBinary, 273
HMetadataGenericBinary, 273
HMetadataGenericBool, 274
BuildMetadataString, 275
HMetadataGenericBool, 274, 275
HMetadataGenericByte, 275
BuildMetadataString, 276
HMetadataGenericByte, 276
HMetadataGenericDouble, 276
BuildMetadataString, 277
HMetadataGenericDouble, 277
HMetadataGenericInt, 278
BuildMetadataString, 279
HMetadataGenericInt, 278, 279
HMetadataGenericInt64, 279
BuildMetadataString, 280
HMetadataGenericInt64, 280
HMetadataGenericShort, 280
BuildMetadataString, 281
HMetadataGenericShort, 281
HMetadataGenericString, 282
BuildMetadataString, 283
HMetadataGenericString, 282, 283
HMetadataGenericUInt, 283
BuildMetadataString, 284
HMetadataGenericUInt, 284
HMetadataGenericUInt64, 284
BuildMetadataString, 285
HMetadataGenericUInt64, 285
HMetadataGenericUShort, 286
BuildMetadataString, 287
HMetadataGenericUShort, 286, 287
HMetadataInfo, 287
GetPacketSize, 288
HMetadataInfo, 288
WritePacketTo, 288
HMetadataManager, 289
GetMetadataTypeByID, 290
GetMetadataTypeByIndex, 290
GetMetadataTypeCount, 290
RebuildMetadataString, 290
RegisterGenericTypes, 289
RegisterMetadataType, 289
RegisterNorPixTypes, 290
HMetadataNorPixCameraInfo, 291
BuildMetadataString, 292
GetAperture, 292
GetExposure, 292
GetFocalLength, 292
GetGain, 293
GetGamma, 293
HMetadataNorPixCameraInfo, 292
SetAperture, 293
SetExposure, 293
SetFocalLength, 293
SetGain, 293
SetGamma, 294
HMetadataNorPixDMIExtended, 294
BuildMetadataString, 295
HMetadataNorPixDMIExtended, 294, 295
HMetadataNorPixDMIFBExtended, 295
BuildMetadataString, 296
HMetadataNorPixDMIFBExtended, 296
HMetadataNorPixGPS, 296
BuildMetadataString, 297
HMetadataNorPixGPS, 297
HMetadataNorPixGPSExtended, 297
BuildMetadataString, 301
HMetadataNorPixGPSExtended, 299–301
HMetadataNorPixGRTHeader, 301
BuildMetadataString, 303
HMetadataNorPixGRTHeader, 302, 303
HMetadataNorPixImageStatistics, 303
BuildMetadataString, 305
HMetadataNorPixImageStatistics, 304, 305
HMetadataNorPixInputLineState, 305
BuildMetadataString, 306
GetDeviceName, 307
GetInputLevel, 307
GetLineName, 307
HMetadataNorPixInputLineState, 306
HMetadataNorPixLTC, 308

BuildMetadataString, 309
GetLTC, 309
HMetadataNorPixLTC, 308, 309
HMetadataNorPixLidar, 307
BuildMetadataString, 308
HMetadataNorPixTime64, 309
BuildMetadataString, 310
HMetadataNorPixTime64, 310
HMotionDetection, 310
AbsDiffDetect, 315, 321
AbsMaxDiffDetect, 317, 322
AdaptiveEdgeDetect, 318, 323
ConvertToGray, 315, 321
EdgeDetect, 317, 322
GetAbsoluteThresholdValue, 313, 320
GetLastError, 312, 319
GetMotionDetectionMode, 312, 319
GetRoi, 315, 321
GetSobelDetectionMode, 313, 320
GetThresholdValue, 313, 320
MeanGrayDetect, 315, 321
MoveTowards, 319, 324
OverlayImage, 318, 323
SetAbsoluteThresholdValue, 313, 320
SetErrorOutput, 312, 319
SetMotionDetectionMode, 312, 319
SetRoi, 313, 320
SetSobelDetectionMode, 313, 319
SetThresholdValue, 313, 320
HRotation, 341
GetCenterCoordinate, 343, 346
GetDestinationImageSize, 343, 346
GetInterpolateMode, 345, 347
GetLastError, 342, 346
GetRotateAngle, 343, 347
GetSmoothEdge, 345, 347
IsImageFormatSupported, 345, 347
Rotate, 345, 348
SetCenterCoordinate, 343, 346
SetDestinationImageSize, 342, 346
SetErrorOutput, 342, 346
SetInterpolateMode, 343, 347
SetRotateAngle, 343, 347
SetSmoothEdge, 345, 347
HSequence, 348
Close, 354
Create, 353
DumpHeader, 356
EnableGPUJpegCompressionBayerInterp, 370
EnableGPUJpegCompressionColorprcessingLUT, 371
EnableMetadata, 359
GetAllocatedFrames, 358
GetAutoRecoveryIndexStream, 367
GetBayerPattern, 369
GetCayerCompressionType, 366
GetCompressSeqIndexType, 366
GetCompressionBayerPattern, 366
GetDescription, 367
GetDescriptionFormat, 368
GetFrameRate, 356
GetImageBitDepth, 364
GetImageBitDepthReal, 364
GetImageFormat, 365
GetImageHeight, 364
GetImageSizeBytes, 364
GetImageWidth, 364
GetIndexAt, 358
GetIndexNear, 358
GetJpegDecodeType, 370
GetJpegEncodeType, 370
GetLastError, 353
GetMaxPlaybackInterval, 362
GetNextPlaybackFrameTime, 364
GetOldestFrameIndex, 372
GetPlaybackFrameSkip, 362
GetPlaybackPosition, 360
GetPlaybackRange, 363
GetPlaybackSpeed, 361
GetPreAllocatedImages, 359
GetRecordingPosition, 356
GetReferenceFrame, 369
GetReferenceTime, 370
GetSequenceCompression, 365
GetTrueImageSize, 365
HasLTC, 371
HasMarkers, 371
HasReferenceTime, 369
IsBayer, 368
IsPlayFixedRate, 360
IsPlaybackLagging, 361
IsPlaying, 362
IsRAMSequence, 354
IsSequenceOpen, 354
Open, 354
Play, 360
PlayFixedRate, 360
PreAllocateImages, 359
Read, 355
ReadTimestamp, 358
RecoverIndexStream, 367
ResynchronizePlayback, 363
SetAutoRecoveryIndexStream, 367
SetBayerPattern, 368
SetCallbacks, 353
SetCayerCompressionType, 366
SetCompressSeqIndexType, 366
SetCompressionBayerPatternType, 366
SetDescription, 368
SetDescriptionFormat, 368
SetErrorOutput, 353
SetFrameRate, 356
SetJpegDecoderType, 370
SetJpegEncoderType, 370
SetMaxPlaybackInterval, 362
SetOldestFrameIndex, 372

SetPlaybackFrameSkip, 361
SetPlaybackPosition, 359
SetPlaybackRange, 362
SetPlaybackSpeed, 361
SetRecordingPosition, 355
SetReferenceFrame, 369
SetReferenceTime, 369
SetTimeOffset, 372
SetVideoEncoderType, 365
SetVideoSequenceParameter, 365
Sort, 371
Stop, 361
SupportRewrite, 356
Truncate, 371
UpdateGPUJpegCompressionLUT, 371
Write, 354
HSequencerCallbacks, 372
OnEndOfPlaybackRange, 374
OnEndOfSequence, 373
OnImageRead, 373
OnImageWrite, 373
OnPlaybackStopped, 374
OnQueryContinuePlayback, 373
HStreamDeliver, 374
AddVirtualURL, 375
DeliverNewAudio, 375
DeliverNewFrame, 375
RemoveVirtualURL, 375
HStreamServer, 376
RegisterSourceUrl, 377
SetCallback, 378
SetHttpPort, 377
SetRtspCredential, 378
SetRtspPort, 377
StartServer, 378
UnRegisterSourceUrl, 377
HTTPRequest_Callback, 389
NewHttpRequest, 389
HTcpCallbacks, 378
OnClientConnected, 379
OnClientDisconnected, 379
OnReceiveData, 379
HTcpClient, 379
ConnectTo, 381
Disconnect, 381
GetLastError, 380
GetPeerAddress, 382
GetSocket, 382
IsConnected, 381
Send, 382
SetCallback, 380
SetErrorOutput, 380
HTcpServer, 383
Broadcast, 387
CloseConnection, 385
CloseServer, 385
GetAddress, 386
GetConnectionCount, 385
GetFirstSocket, 387
GetLastError, 384
GetNextSocket, 387
GetPeerAddress, 388
GetSocket, 386
IsServerOpen, 385
OpenServer, 384
Send, 387
SetCallback, 384
SetErrorOutput, 384
HTimeSourceSyncStatus
HEnums.h, 419
HUdpCallbacks, 390
OnReceiveData, 390
HUdpClient, 390
Connect, 391
GetLastError, 391
IsConnected, 391
SendTo, 392
SendToEx, 392
SetCallback, 391
SetErrorOutput, 391
HUdpServer, 392
GetIPAddress, 394
GetLastError, 393
IsOpened, 393
Open, 394
SendTo, 394
SendToEx, 395
SetCallback, 393
SetErrorOutput, 393
HVideoOutputDriver, 395
DisableVideoOutput, 396
EnableVideoOutput, 396
IsVideoOutputAvailable, 396
SendImageForVideoOutput, 396
HViewer, 396
GetLastError, 397
GetZoom, 398
SetErrorOutput, 397
SetZoom, 397
ShowImage, 398
ShowImageOnDC, 398
ShowScaledImage, 398
HWhiteBalance, 399
ApplyColorBalance, 403
ComputeColorBalance, 401, 403
GetAlgorithm, 400
GetAutoColorBalanceMode, 401
GetColorBalance, 400
GetColorBalanceApplied, 401
GetLastError, 400
SetAlgorithm, 401
SetAutoColorBalanceMode, 401
SetColorBalance, 400
SetColorBalanceApplied, 401
SetErrorOutput, 400
HasImageDecoded

HImage, 191
HasLTC
HSequence, 371
HasMarkers
HSequence, 371
HasReferenceTime
HSequence, 369
HermesUtils.h, 431
Http_Request, 388
Http_Request::mg_header, 404

I/O devices, 17
Image Files, 16
Image Processing, 11
InitializeDaqDevice
HIODriver, 251
InitializeDeviceInput
HIODriver, 246
InitializeDeviceOutput
HIODriver, 248
InitializePulseGenerator
HIODriver, 250
InputDevicePresent
HIODriver, 245
Is3DLookupTable
HImageColorProcessing, 204
IsAMDH264Present
Utilities, 22
IsAMDOpenCLEncodingEnabled
HGPUDeviceManager, 148
IsAudioInputCapturing
HAudioDriverASIO, 52
HAudioDriverDirectX, 61
IsAudioOutputPlaying
HAudioDriverASIO, 56
HAudioDriverDirectX, 65
IsAutoColorBalanceEnabled
HImageColorProcessing, 202
IsBGRxConversionEnabled
HImageColorProcessing, 200
IsBayer
HSequence, 368
IsBayerConversionEnabled
HImageColorProcessing, 197
IsBinaryBitmap
HBitmapOverlay, 110
IsCOMPortAvailable
HGrabModule, 171
IsColorProcessingHQConversionEnabled
HImageColorProcessing, 200
IsConnected
HTcpClient, 381
HUdpClient, 391
IsDummyDataSet
HDaqDataSet, 117, 120
IsExporting
HAviExporter, 90
IsFormatMono
Utilities, 19
IsFormatSupported
HJpegCompression, 259
IsFormatToF
Utilities, 19
IsGPSSupported
HGrabModule, 171
IsImageFormatSupported
HRotation, 345, 347
IsKeepImageRatio
HImageResize, 241
IsLUTProcessEnabled
HImageColorProcessing, 204
IsLiveRecording
HAviFile, 101
IsLooping
HAviFile, 103
IsMatrixColorCorrectionEnabled
HImageColorProcessing, 200
IsMonochromeConversionEnabled
HImageColorProcessing, 209
IsNvidiaCudaEncodingEnabled
HGPUDeviceManager, 148
IsNvidiaGPUHEVCPresent
Utilities, 22
IsNvidiaH264Present
Utilities, 22
IsNvidiaOpenCLEncodingEnabled
HGPUDeviceManager, 147
IsOpened
HUdpServer, 393
IsPaused
HAviFile, 102
IsPlayFixedRate
HSequence, 360
IsPlaybackLagging
HSequence, 361
IsPlaying
HAudioPlayer, 78
HAviFile, 102
HDaqFile, 128, 134
HMP4File, 337
HSequence, 362
IsRAMSequence
HSequence, 354
IsRecording
HAudioRecorder, 84
IsSequenceOpen
HSequence, 354
IsServerOpen
HTcpServer, 385
IsStopped
HAviFile, 102
IsStreaming
HGrabModule, 161
IsSuggestPushAudio
HMP4File, 332
IsSuggestPushVideo
HMP4File, 332

IsSupportAdvanceSettings
HGrabModule, 174
IsSupportPanTilt
HGrabModule, 182
IsSupportZoom
HGrabModule, 183
IsVideoOutputAvailable
HVideoOutputDriver, 396

Load
HGrabModule, 159
Load1DPreDefinedLUT
HImageColorProcessing, 208
LoadNpxTextLutFile
HImageColorProcessing, 208

MOV File, 14
MeanGrayDetect
HMotionDetection, 315, 321
Merge
HImageMerge, 225
Metadata, 23
MetadataID.h, 432
Mirror
HImageMirror, 227
Monitor
HIODriver, 247
MoveTo
HAudioPlayer, 79
MoveTowards
HMotionDetection, 319, 324
MultipleMerge
HImageMerge, 225

Network, 26
NewHttpRequest
HTTPRequest_Callback, 389
NewLiveFrame
HFPSMonitor, 139
NorPix Metadata, 25

OnAfterQueuingGrab
HGrabCallbacks, 151
OnAfterStartStreaming
HGrabCallbacks, 151
OnAfterStopStreaming
HGrabCallbacks, 152
OnAudioSourceEvent
CAudioPlaybackCallback, 32
OnAviEndPlayback
HAviFileCallbacks, 107
OnAviImageRead
HAviFileCallbacks, 107
OnBeforeImageReceived
HGrabCallbacks, 150
OnBeforeQueuingGrab
HGrabCallbacks, 151
OnBeforeStartStreaming
HGrabCallbacks, 151
OnBeforeStopStreaming
HGrabCallbacks, 151
OnBufferReceived
CAudioRecorderCallback, 33
OnBufferSaved
CAudioRecorderCallback, 33
OnClientConnected
HTcpCallbacks, 379
OnClientDisconnected
HTcpCallbacks, 379
OnDaqDeviceStatusChanged
CDaqSupportCallback, 36, 37
OnDaqPlaybackUpdate
HDaqFileCallbacks, 136, 137
OnDataReceived
CAudioCaptureCallback, 31
CDaqSupportCallback, 36
OnEndOfPlaybackRange
HSequencerCallbacks, 374
OnEndOfSequence
HSequencerCallbacks, 373
OnEndPlayback
CAudioPlayerCallback, 33
OnGPSSentencesReceived
HGrabCallbacks, 152
OnImageRead
HSequencerCallbacks, 373
OnImageReceived
HGrabCallbacks, 150
OnImageReleased
HGrabCallbacks, 150
OnImageWrite
HSequencerCallbacks, 373
OnNewFrameRate
HFPSMonitorCallbacks, 141
OnNotification
HGrabCallbacks, 152
OnPlayBuffer
CAudioPlayerCallback, 33
OnPlaybackStopped
HSequencerCallbacks, 374
OnPrepareCompress
HDeliverCallback, 137
OnQueryContinuePlayback
HSequencerCallbacks, 373
OnReadNewData
CAudioPlaybackCallback, 32
OnReceiveData
HTcpCallbacks, 379
HUdpCallbacks, 390
OneShotAdjustment
HGrabModule, 169
Open
HAACFile, 40
HAudioFile, 69
HDaqFile, 123, 130
HMP3File, 326
HMP4File, 333

HSequence, 354
HUdpServer, 394
OpenAudioInputDevice
HAudioDriverASIO, 49
HAudioDriverDirectX, 59
OpenAudioInputHardwareSettingsDlg
HAudioDriverASIO, 50
OpenAudioOutputDevice
HAudioDriverASIO, 53
HAudioDriverDirectX, 63
OpenAudioOutputHardwareSettingsDlg
HAudioDriverASIO, 54
OpenAviFile
HAviFile, 97
OpenChannels
HAudioRecorder, 83
OpenPlayer
HAudioPlayer, 77, 78
OpenServer
HTcpServer, 384
operator=
HMetadata, 271
OutputDevicePresent
HIODriver, 248
OverlayImage
HMotionDetection, 318, 323

Pause
HAviFile, 102
Play
HAudioPlayer, 78
HAviFile, 101
HMP4File, 336
HSequence, 360
PlayFixedRate
HMP4File, 336
HSequence, 360
PreAllocateImages
HSequence, 359
PreparePlayback
HAudioFile, 70
PrepareWrite
HMP4File, 332
HMPEGTsFile, 339
PulseGeneratorPresent
HIODriver, 250
PushImageToQueue
HLineScanManager, 263, 266
PushLiveAudioData
HAudioPlayer, 78

ReSample
HImageResize, 242
ReSizeAdvanceSettingsWindow
HGrabModule, 174
Read
HAudioFile, 70, 71
HDaqFile, 124, 125, 130, 131
HMP4File, 334, 335
HSequence, 355
ReadAVIFrameRate
Utilities, 21
ReadAudio
HAACFile, 41
HMP3File, 327
HMP4File, 337
ReadImage
HAviFile, 102
ReadTimestamp
HSequence, 358
ReallocateAudio
HAudioData, 44
ReallocateCompressedImage
HImage, 191
ReallocateImage
HImage, 191
RebuildMetadataString
HMetadataManager, 290
Receive
HGrabModule, 172
RecoverIndexStream
HSequence, 367
RegisterGenericTypes
HMetadataManager, 289
RegisterMetadataType
HMetadataManager, 289
RegisterNorPixTypes
HMetadataManager, 290
RegisterSourceUrl
HStreamServer, 377
RelativeToAbsolute
Utilities, 20
RemoveMetadata
HDaqFile, 128, 135
RemoveThirdPartLutFile
HImageColorProcessing, 206
RemoveVirtualURL
HStreamDeliver, 375
ResetCaptureBufferUsage
HAudioDriverASIO, 52
HAudioDriverDirectX, 62
Resize
HImageResize, 242
ResynchronizePlayback
HDaqFile, 128, 135
HSequence, 363
RollingFrameAverage
HFrameAverage, 144, 145
Rotate
HRotation, 345, 348

SEQ File, 15
Send
HTcpClient, 382
HTcpServer, 387
SendImageForVideoOutput
HVideoOutputDriver, 396
SendTo

HUdpClient, 392
HUdpServer, 394
SendToEx
HUdpClient, 392
HUdpServer, 395
SetASCSettingsParam
HImageColorProcessing, 204
SetAbsoluteThresholdValue
HMotionDetection, 313, 320
SetAlgorithm
HWhiteBalance, 401
SetAlpha
HImageOverlay, 230
SetAperture
HMetadataNorPixCameraInfo, 293
SetAudioCodec
HAviExporter, 89
HAviFile, 100
SetAudioDriver
HAudioRecorder, 81
SetAudioFile
HAudioRecorder, 82, 83
SetAudioInputFormat
HAudioDriverDirectX, 60
SetAudioInputTimeSourceDLL
HAudioDriverASIO, 49
HAudioDriverDirectX, 59
SetAudioOutputFormat
HAudioDriverDirectX, 64
SetAudioParam
HAACFile, 39
HMP3File, 326
HMP4File, 331
HMPEGTsFile, 339
SetAudioSource
HAviExporter, 88
HAviFile, 99
SetAuthor
HAviFile, 106
SetAutoColorBalanceAlgorithm
HImageColorProcessing, 203
SetAutoColorBalanceMode
HWhiteBalance, 401
SetAutoRecoveryIndexStream
HSequence, 367
SetAutoWhiteBalanceROI
HImageColorProcessing, 203
SetAutomaticAdjustment
HGrabModule, 169
SetBayerPattern
HSequence, 368
SetBinning
HGrabModule, 164
SetBitmapFile
HBitmapOverlay, 109
SetBoolFeatureValue
HGrabModule, 180
SetBufferCount
HAudioRecorder, 85
HGrabModule, 182
SetCacheLUTFileName
HImageColorProcessing, 205
SetCalcMode
HImageColorProcessing, 199
SetCallback
HAudioRecorder, 81
HStreamServer, 378
HTcpClient, 380
HTcpServer, 384
HUdpClient, 391
HUdpServer, 393
SetCallbacks
HAviFile, 94
HFPSMonitor, 141
HGrabModule, 161
HMP4File, 333
HSequence, 353
SetCaptureBufferCount
HAudioDriverASIO, 52
HAudioDriverDirectX, 62
SetCayerCompressionType
HSequence, 366
SetCenterCoordinate
HRotation, 343, 346
SetCineFormCodecHDParams
HAviFile, 101
SetCineFormCodecRawParams
HAviFile, 100
SetColorBalance
HWhiteBalance, 400
SetColorBalanceApplied
HWhiteBalance, 401
SetColorClipLevel
HImageColorProcessing, 206
SetCompressSeqIndexType
HSequence, 366
SetCompressionBayerPatternType
HSequence, 366
SetCopyright
HAviFile, 105
SetCorrectionMatrix
HImageColorProcessing, 201
SetCreationDateTime
HAviFile, 105
SetCrosshairThickness
HImageOverlay, 231
SetCurrentValues
HGrabModule, 166
SetDaqDataSet
HDaqDataSet, 116, 119
SetDaqDeviceEnumParameterValue
HIODriver, 256
SetDaqDeviceFloatParameterValue
HIODriver, 257
SetDaqDeviceIntParameterValue
HIODriver, 257

SetDateTimeFormat
HImageExporter, 223
SetDateTimeOriginal
HAviFile, 106
SetDefaultGrabber
HGrabModule, 181
SetDescription
HSequence, 368
SetDescriptionFormat
HSequence, 368
SetDestinationImageSize
HRotation, 342, 346
SetDestinationSize
HImageResize, 240
SetDirection
HLineScanManager, 263, 266
SetDisplayColorForBinaryBitmap
HBitmapOverlay, 110
SetDoubleAdjustment
HGrabModule, 168
SetDoubleFeatureValue
HGrabModule, 179
SetDummyDataSet
HDaqDataSet, 116, 119
SetEnumFeatureValue
HGrabModule, 177
SetErrorOutput
HAACFile, 38
HAudioDriverASIO, 48
HAudioDriverDirectX, 58
HAudioFile, 68
HAudioPlayer, 74
HAudioRecorder, 80
HAviExporter, 87
HAviFile, 94
HBitmapOverlay, 109
HDaqFile, 123, 129
HFPSMonitor, 139
HFrameAverage, 143, 144
HGPUDeviceManager, 147
HGrabModule, 158
HGrayConverter, 185, 186
HImageColorProcessing, 196
HImageConverter, 210
HImageExporter, 213
HImageMerge, 224
HImageMirror, 226
HImageOverlay, 229
HImageResize, 240
HJpegCompression, 259
HMP3File, 325
HMP4File, 330
HMPEGTsFile, 338
HMotionDetection, 312, 319
HRotation, 342, 346
HSequence, 353
HTcpClient, 380
HTcpServer, 384
HUdpClient, 391
HUdpServer, 393
HViewer, 397
HWhiteBalance, 400
SetExposure
HMetadataNorPixCameraInfo, 293
SetFileSource
HAudioPlayer, 74, 76
SetFocalLength
HMetadataNorPixCameraInfo, 293
SetFrameRate
HSequence, 356
SetGain
HMetadataNorPixCameraInfo, 293
SetGamma
HMetadataNorPixCameraInfo, 294
SetHttpPort
HStreamServer, 377
SetIntegerAdjustment
HGrabModule, 168
SetIntegerFeatureValue
HGrabModule, 178
SetInterpolateMode
HRotation, 343, 347
SetInterpolationMode
HImageResize, 241
SetJPEGQuality
HImageExporter, 220
SetJpegDecoderType
HSequence, 370
SetJpegEncoderType
HImageExporter, 221
HSequence, 370
SetJpegGPUCompressionBayerPatternType
HImageExporter, 221
SetJpegGPUEncoderType
HImageExporter, 221
SetKeepImageRatio
HImageResize, 241
SetLineColor
HImageOverlay, 232
SetLineCount
HLineScanManager, 263, 266
SetLineStyle
HImageOverlay, 232
SetLineThickness
HImageOverlay, 232
SetLiveRecording
HAviFile, 101
SetLiveSource
HAudioPlayer, 76
SetLooping
HAviFile, 103
SetManualColorBalance
HImageColorProcessing, 202
SetMaxPlaybackInterval
HSequence, 362
SetMergeMode

HImageMerge, 224
SetMetadata
HDaqFile, 129, 135
SetMetadataSize
HDaqFile, 128, 135
SetMetadataString
HMetadata, 272
SetMirrorMode
HImageMirror, 226
SetMonoPseudoColorMode
HImageColorProcessing, 207
SetMotionDetectionMode
HMotionDetection, 312, 319
SetOldestFrameIndex
HSequence, 372
SetOutputLevel
HIODriver, 250
SetOutputVolume
HAudioDriverDirectX, 65
SetOverlayColor
HImageOverlay, 229
SetOverlayPosition
HBitmapOverlay, 110
SetPanTilt
HGrabModule, 183
SetPatternOrigin
HImageColorProcessing, 197
SetPlaybackFormat
HAviFile, 104
SetPlaybackFrameRate
HMP4File, 336
SetPlaybackFrameSkip
HSequence, 361
SetPlaybackPosition
HDaqFile, 127, 134
HMP4File, 335
HSequence, 359
SetPlaybackRange
HSequence, 362
SetPlaybackRate
HAviFile, 104
SetPlaybackSpeed
HSequence, 361
SetPositionByTimeStamp
HAudioFile, 73
SetROI
HGrabModule, 164
SetReadingPosition
HAACFile, 41
HAudioFile, 71
HMP3File, 328
SetRecordingPosition
HSequence, 355
SetReferenceFrame
HSequence, 369
SetReferenceFrameIndex
HDaqDataSet, 115, 118
SetReferenceTime
HSequence, 369
SetRoi
HMotionDetection, 313, 320
SetRotateAngle
HRotation, 343, 347
SetRtspCredential
HStreamServer, 378
SetRtspPort
HStreamServer, 377
SetSampleFactor
HImageColorProcessing, 199
SetSeparator
HImageExporter, 223
SetSmoothEdge
HRotation, 345, 347
SetSobelDetectionMode
HMotionDetection, 313, 319
SetSoftwareName
HAviFile, 106
SetSourceRoi
HImageResize, 240
SetStartTimeStamp
HAudioFile, 72
SetTextBackgroundColor
HImageOverlay, 230
SetTextOverlayAlign
HImageOverlay, 230
SetTextOverlayFont
HImageOverlay, 231
SetTextOverlaySize
HImageOverlay, 231
SetThresholdValue
HMotionDetection, 313, 320
SetTimeOffset
HSequence, 372
SetTimeSourceDLL
HIODriver, 252
SetTimeStamp
HAudioData, 45
HDaqDataSet, 116, 119
SetTimestamp
HImage, 192
SetTitle
HAviFile, 105
SetTransparentcyColor
HBitmapOverlay, 109
SetVideoCodec
HAviExporter, 88
HAviFile, 98
SetVideoEncoderType
HSequence, 365
SetVideoParam
HMP4File, 331
HMPEGTsFile, 339
SetVideoSequenceParameter
HSequence, 365
SetZoom
HGrabModule, 184

HViewer, 397
ShowAdvanceSettingsWindow
HGrabModule, 175
ShowAudioCodecSettings
HAviExporter, 89
HAviFile, 100
ShowImage
HViewer, 398
ShowImageOnDC
HViewer, 398
ShowProperties
HGrabModule, 162
ShowScaledImage
HViewer, 398
ShowVideoCodecSettings
HAviExporter, 88
HAviFile, 98
Sort
HSequence, 371
StandardFrameAverage
HFrameAverage, 143, 145
StartAudioInputCapture
HAudioDriverASIO, 51
HAudioDriverDirectX, 61
StartAudioOutputPlayback
HAudioDriverASIO, 55
HAudioDriverDirectX, 65
StartBench
HFPSMonitor, 140
StartCapture
HIODriver, 253
StartLiveCheck
HFPSMonitor, 139
StartPlayback
HDaqFile, 127, 134
StartPulseGenerator
HIODriver, 250
StartQueryMode
HGrabModule, 180
StartRecording
HAudioRecorder, 84
StartServer
HStreamServer, 378
StartStreaming
HGrabModule, 161
Stitch
HImageMerge, 225
Stop
HAudioPlayer, 78
HAviFile, 102
HMP4File, 336
HSequence, 361
StopAudioInputCapture
HAudioDriverASIO, 51
HAudioDriverDirectX, 61
StopAudioOutputPlayback
HAudioDriverASIO, 55
HAudioDriverDirectX, 65
StopCapture
HIODriver, 254
StopPlayback
HDaqFile, 127, 134
StopPulseGenerator
HIODriver, 251
StopReadingGPSData
HGrabModule, 171
StopRecording
HAudioRecorder, 84
HAviFile, 97
StopStreaming
HGrabModule, 161
SuggestBottomRightPosition
HBitmapOverlay, 112
SuggestCenterPosition
HBitmapOverlay, 110
SupportRewrite
HSequence, 356

TCP Network, 28
Transmit
HGrabModule, 171
Truncate
HSequence, 371
TryToLock
HLocker, 269

UDP Network, 27
UnInitializeDaqDevice
HIODriver, 252
UnInitializeDeviceInput
HIODriver, 246
UnInitializeDeviceOutput
HIODriver, 249
UnInitializePulseGenerator
HIODriver, 250
UnRegisterSourceUrl
HStreamServer, 377
Unload
HGrabModule, 160
UpdateGPUJpegCompressionLUT
HImageExporter, 222
HSequence, 371
UseSpecialCodec
HAviExporter, 89
Utilities, 18
AbsoluteToRelative, 19
AddTimestampOffset, 21
CompareTimestamps, 21
ComputeTimeDifference, 20
CreateDirectoryRecursive, 21
HImageCompressionString, 19
HImageFormatString, 19
IsAMDH264Present, 22
IsFormatMono, 19
IsFormatToF, 19
IsNvidiaGPUHEVCPresent, 22
IsNvidiaH264Present, 22

ReadAVIFrameRate, 21
RelativeToAbsolute, 20
WriteAVIFrameRate, 22

Video, 8
Viewer, 29

WantContinuousDataRead
CDaqSupportCallback, 35, 36
Write
HAudioFile, 70
HDaqFile, 123, 130
HSequence, 354
WriteAVIFrameRate
Utilities, 22
WriteAlignedMultilinesTextOverlay
HImageOverlay, 234
WriteAlignedTextOverlay
HImageOverlay, 233
WriteAngleOverlay
HImageOverlay, 237
WriteArrowOverlay
HImageOverlay, 236
WriteBezierOverlay
HImageOverlay, 238
WriteBitmapOverlay
HBitmapOverlay, 112
WriteCircleOverlay
HImageOverlay, 237
WriteCrosshair
HImageOverlay, 235
WriteCustomOverlay
HImageOverlay, 239
WriteEllipseOverlay
HImageOverlay, 236
WriteFilledEllipseOverlay
HImageOverlay, 237
WriteFilledPolyOverlay
HImageOverlay, 238
WriteFilledRectOverlay
HImageOverlay, 236
WriteFlatEndsLineOverlay
HImageOverlay, 235
WriteLineOverlay
HImageOverlay, 235
WriteMultilineTextOverlay
HImageOverlay, 234
WritePacketTo
HMetadata, 272
HMetadataInfo, 288
WritePolyOverlay
HImageOverlay, 238
WriteRectOverlay
HImageOverlay, 236
WriteRoundEndsLineOverlay
HImageOverlay, 235
WriteTextOverlay
HImageOverlay, 234

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich, Germany
+49 89 780 2970, info@skyblue.de
www.skyblue.de

In Great Britain:
Zerif Technologies Ltd.
Woodberry House, 2 Woodberry Grove
Finchley, London N12 0DR
+44 115 855 7883, info@zerif.co.uk
www.zerif.co.uk