



ASX

Version 4.18.08

Generated by Doxygen 1.7.3

Fri Sep 2 2016 10:32:45

Contents

1	Main Page	1
1.1	ASX Reference Manual	1
1.2	Introduction	1
1.3	Installation and Usage	2
1.3.1	Windows XP and Windows 7	2
1.3.2	Linux	2
1.4	Compiling applications that use the ASX interface	3
1.5	Debugging ASX calls under Windows	3
1.6	ASX Object Model	3
1.7	Coding Conventions	5
2	Porting Applications from PCXTools	7
2.1	Global	7
2.1.1	TOOLRegister(),ToolUnregister()	7
2.1.2	TOOLSetOEMAuthorizationDll	7
2.1.3	TOOLGetLastError, TOOLGetErrorString	7
2.2	Configuration	8
2.2.1	TOOLGetVersions	8
2.2.2	TOOLHowManyTotalOutputs	8
2.2.3	TOOLHowManyTotalInputs	8
2.2.4	TOOLGetBoardInfos	8
2.2.5	TOOLGetBoardName	8
2.3	Pipe	8
2.3.1	TOOLAllocatePipe	8
2.3.2	TOOLAllocatePipeEx	8
2.3.3	TOOLReleasePipe	8
2.3.4	TOOLPipeStart	8
2.3.5	TOOLPipeStop	8
2.3.6	TOOLPipeSetLevel	8
2.3.7	TOOLPipeSetMonitoringMute	8
2.3.8	TOOLPipeSetClock	8
2.3.9	TOOLPipeDefineDigitalInput	8
2.3.10	TOOLPipeSetStretch	8
2.3.11	TOOLPipeGetVuMeter	8
2.3.12	TOOLPipeGetClock	8
2.3.13	TOOLPipeGetPosition	8
2.4	Stream	8
2.5	Offline	9
2.6	Misc	9

3	Deprecated List	11
4	Module Index	13
5	Data Structure Index	15
5.1	Data Structures	15
6	File Index	17
7	Module Documentation	19
7.1	SubSystem types	19
7.1.1	Detailed Description	20
7.1.2	Define Documentation	20
7.1.2.1	ASX_SYSTEM_TYPE_ALSA	20
7.1.2.2	ASX_SYSTEM_TYPE_ANY	20
7.1.2.3	ASX_SYSTEM_TYPE_ASIO	20
7.1.2.4	ASX_SYSTEM_TYPE_AVB_1722_1	20
7.1.2.5	ASX_SYSTEM_TYPE_COUNT	20
7.1.2.6	ASX_SYSTEM_TYPE_DIRECTX	21
7.1.2.7	ASX_SYSTEM_TYPE_DUMMY	21
7.1.2.8	ASX_SYSTEM_TYPE_HPI	21
7.1.2.9	ASX_SYSTEM_TYPE_HPIUDP	21
7.1.2.10	ASX_SYSTEM_TYPE_PCXTOOLS	21
7.1.2.11	ASX_SYSTEM_TYPE_PORTAUDIO	21
7.1.2.12	ASX_SYSTEM_TYPE_SNMP	21
7.1.2.13	ASX_SYSTEM_TYPE_WAVE	22
7.2	System functions	22
7.2.1	Function Documentation	23
7.2.1.1	ASX_System_Create	23
7.2.1.2	ASX_System_CreateSubSystem	23
7.2.1.3	ASX_System_Delete	24
7.2.1.4	ASX_System_GetAdapter	24
7.2.1.5	ASX_System_GetAdapterCount	25
7.2.1.6	ASX_System_GetCobranetAutoassignParms	25
7.2.1.7	ASX_System_GetMessageLogging	26
7.2.1.8	ASX_System_GetName	26
7.2.1.9	ASX_System_GetVersion	27
7.2.1.10	ASX_System_RegisterErrorCallback	28
7.2.1.11	ASX_System_SetAvdeccInterface	28
7.2.1.12	ASX_System_SetCobranetAutoassignParms	28
7.2.1.13	ASX_System_SetHostNetworkInterface	29
7.2.1.14	ASX_System_SetMessageLogging	29
7.2.1.15	ASX_System_SupportsSubSystem	29
7.3	Handle functions	30
7.3.1	Detailed Description	30
7.3.2	Function Documentation	30
7.3.2.1	ASX_Handle_GetType	30
7.4	Error functions	30
7.4.1	Detailed Description	31
7.4.2	Function Documentation	31

7.4.2.1	ASX_Error_Clear	31
7.4.2.2	ASX_Error_GetLast	31
7.4.2.3	ASX_Error_GetLastString	32
7.5	Adapter functions	33
7.5.1	Detailed Description	34
7.5.2	Function Documentation	35
7.5.2.1	ASX_Adapter_CheckSubSystems	35
7.5.2.2	ASX_Adapter_EnumerateMode	35
7.5.2.3	ASX_Adapter_EnumerateProperty	35
7.5.2.4	ASX_Adapter_GetAVDECCController	36
7.5.2.5	ASX_Adapter_GetAVDECCSystem	36
7.5.2.6	ASX_Adapter_GetDspUtilization	36
7.5.2.7	ASX_Adapter_GetFirmwareRevision	37
7.5.2.8	ASX_Adapter_GetHardwareRevision	37
7.5.2.9	ASX_Adapter_GetIndex	38
7.5.2.10	ASX_Adapter_GetIpAddress	39
7.5.2.11	ASX_Adapter_GetMacAddress	39
7.5.2.12	ASX_Adapter_GetMixer	39
7.5.2.13	ASX_Adapter_GetMode	40
7.5.2.14	ASX_Adapter_GetName	40
7.5.2.15	ASX_Adapter_GetNvMemSizeInBytes	41
7.5.2.16	ASX_Adapter_GetSerialNumber	41
7.5.2.17	ASX_Adapter_ReadNvMem	42
7.5.2.18	ASX_Adapter_ReadProperty	42
7.5.2.19	ASX_Adapter_SetMode	43
7.5.2.20	ASX_Adapter_WriteNvMem	43
7.5.2.21	ASX_Adapter_WriteProperty	43
7.6	Mixer functions	44
7.6.1	Detailed Description	45
7.6.2	Function Documentation	45
7.6.2.1	ASX_Mixer_GetBlockControlByNodeTypeAndIndex	45
7.6.2.2	ASX_Mixer_GetControl	46
7.6.2.3	ASX_Mixer_GetControlByNode	46
7.6.2.4	ASX_Mixer_GetControlByNodeTypeAndIndex	47
7.6.2.5	ASX_Mixer_GetControlCount	47
7.6.2.6	ASX_Mixer_GetDestinationNode	48
7.6.2.7	ASX_Mixer_GetDestinationNodeCount	48
7.6.2.8	ASX_Mixer_GetNodeByType	48
7.6.2.9	ASX_Mixer_GetNodeTypeCount	49
7.6.2.10	ASX_Mixer_GetSourceNode	49
7.6.2.11	ASX_Mixer_GetSourceNodeCount	50
7.6.2.12	ASX_Mixer_ResetControls	50
7.7	Node functions	51
7.7.1	Detailed Description	51
7.7.2	Function Documentation	52
7.7.2.1	ASX_Mixer_GetNodeIndex	52
7.7.2.2	ASX_Mixer_GetNodeType	52
7.7.2.3	ASX_Node_GetIndex	52
7.7.2.4	ASX_Node_GetLocation	52
7.7.2.5	ASX_Node_GetName	53

7.7.2.6	ASX_Node_GetSubSystem	53
7.7.2.7	ASX_Node_GetType	54
7.8	Control generic functions	54
7.8.1	Detailed Description	54
7.8.2	Function Documentation	55
7.8.2.1	ASX_Control_GetDestinationNode	55
7.8.2.2	ASX_Control_GetHpiControl	55
7.8.2.3	ASX_Control_GetSourceNode	55
7.8.2.4	ASX_Control_GetSubSystem	56
7.8.2.5	ASX_Control_GetType	56
7.9	Player control functions	57
7.9.1	Detailed Description	58
7.9.2	Function Documentation	59
7.9.2.1	ASX_Player_Close	59
7.9.2.2	ASX_Player_Format_GetDetails	60
7.9.2.3	ASX_Player_Format_GetString	60
7.9.2.4	ASX_Player_GetFilename	60
7.9.2.5	ASX_Player_GetLoopMode	61
7.9.2.6	ASX_Player_GetPosition	61
7.9.2.7	ASX_Player_GetState	62
7.9.2.8	ASX_Player_GetTimeScale	62
7.9.2.9	ASX_Player_Open	62
7.9.2.10	ASX_Player_OpenPlaylist	63
7.9.2.11	ASX_Player_Pause	64
7.9.2.12	ASX_Player_PlaylistStatus	64
7.9.2.13	ASX_Player_PlaylistWait	65
7.9.2.14	ASX_Player_PreLoad	65
7.9.2.15	ASX_Player_RegisterCallback	66
7.9.2.16	ASX_Player_SetLoopMode	66
7.9.2.17	ASX_Player_SetPosition	66
7.9.2.18	ASX_Player_SetTimeScale	67
7.9.2.19	ASX_Player_Start	67
7.9.2.20	ASX_Player_Stop	67
7.9.2.21	ASX_Player_Wait	68
7.10	Recorder control functions	68
7.10.1	Detailed Description	69
7.10.2	Function Documentation	70
7.10.2.1	ASX_Recorder_Close	70
7.10.2.2	ASX_Recorder_EnumerateFormat	71
7.10.2.3	ASX_Recorder_GetFilename	71
7.10.2.4	ASX_Recorder_GetPosition	71
7.10.2.5	ASX_Recorder_GetState	72
7.10.2.6	ASX_Recorder_Open	72
7.10.2.7	ASX_Recorder_Pause	73
7.10.2.8	ASX_Recorder_Start	73
7.10.2.9	ASX_Recorder_Stop	73
7.11	Meter control functions	74
7.11.1	Detailed Description	74
7.11.2	Function Documentation	74
7.11.2.1	ASX_Meter_GetBallistics	74

7.11.2.2	ASX_Meter_GetChannels	75
7.11.2.3	ASX_Meter_GetPeak	75
7.11.2.4	ASX_Meter_GetRMS	76
7.11.2.5	ASX_Meter_SetBallistics	76
7.12	Volume control functions	77
7.12.1	Detailed Description	78
7.12.2	Function Documentation	78
7.12.2.1	ASX_Volume_GetChannels	78
7.12.2.2	ASX_Volume_GetGain	79
7.12.2.3	ASX_Volume_GetMute	79
7.12.2.4	ASX_Volume_GetRange	79
7.12.2.5	ASX_Volume_SetAutofade	80
7.12.2.6	ASX_Volume_SetGain	80
7.12.2.7	ASX_Volume_SetMute	81
7.13	Level control functions	81
7.13.1	Detailed Description	81
7.13.2	Function Documentation	81
7.13.2.1	ASX_Level_Get	81
7.13.2.2	ASX_Level_GetRange	82
7.13.2.3	ASX_Level_Set	82
7.14	Multiplexer control functions	82
7.14.1	Detailed Description	83
7.14.2	Function Documentation	83
7.14.2.1	ASX_Multiplexer_Enumerate	83
7.14.2.2	ASX_Multiplexer_Get	83
7.14.2.3	ASX_Multiplexer_Set	84
7.15	Channel Mode control functions	84
7.15.1	Detailed Description	85
7.15.2	Function Documentation	85
7.15.2.1	ASX_ChannelMode_Enumerate	85
7.15.2.2	ASX_ChannelMode_Get	85
7.15.2.3	ASX_ChannelMode_Set	86
7.16	Tuner control functions	86
7.16.1	Detailed Description	88
7.16.2	Function Documentation	88
7.16.2.1	ASX_Tuner_EnumerateBand	88
7.16.2.2	ASX_Tuner_EnumerateDeemphasis	89
7.16.2.3	ASX_Tuner_EnumerateHdBlend	89
7.16.2.4	ASX_Tuner_EnumerateProgram	90
7.16.2.5	ASX_Tuner_GetBand	90
7.16.2.6	ASX_Tuner_GetDabAudioInfo	90
7.16.2.7	ASX_Tuner_GetDabAudioServiceCount	91
7.16.2.8	ASX_Tuner_GetDabAudioServiceName	91
7.16.2.9	ASX_Tuner_GetDabMultiplexId	91
7.16.2.10	ASX_Tuner_GetDabMultiplexName	92
7.16.2.11	ASX_Tuner_GetDabServiceId	92
7.16.2.12	ASX_Tuner_GetDeemphasis	92
7.16.2.13	ASX_Tuner_GetDigitalSignalQuality	93
7.16.2.14	ASX_Tuner_GetFirmwareVersion	93
7.16.2.15	ASX_Tuner_GetFrequency	93

7.16.2.16 ASX_Tuner_GetFrequencyRange	94
7.16.2.17 ASX_Tuner_GetGain	94
7.16.2.18 ASX_Tuner_GetGainRange	95
7.16.2.19 ASX_Tuner_GetHdBlend	95
7.16.2.20 ASX_Tuner_GetHdRadioDspVersion	95
7.16.2.21 ASX_Tuner_GetHdRadioSdkVersion	96
7.16.2.22 ASX_Tuner_GetHdRadioSignalQuality	96
7.16.2.23 ASX_Tuner_GetMode	96
7.16.2.24 ASX_Tuner_GetProgram	96
7.16.2.25 ASX_Tuner_GetRawRFLevel	97
7.16.2.26 ASX_Tuner_GetRFLevel	97
7.16.2.27 ASX_Tuner_GetStatus	97
7.16.2.28 ASX_Tuner_SetBand	98
7.16.2.29 ASX_Tuner_SetDabAudioService	98
7.16.2.30 ASX_Tuner_SetDeemphasis	99
7.16.2.31 ASX_Tuner_SetFrequency	99
7.16.2.32 ASX_Tuner_SetGain	99
7.16.2.33 ASX_Tuner_SetHdBlend	100
7.16.2.34 ASX_Tuner_SetMode	100
7.16.2.35 ASX_Tuner_SetProgram	100
7.17 PAD control functions	101
7.17.1 Detailed Description	101
7.17.2 Function Documentation	101
7.17.2.1 ASX_PAD_GetArtist	101
7.17.2.2 ASX_PAD_GetChannelName	102
7.17.2.3 ASX_PAD_GetComment	102
7.17.2.4 ASX_PAD_GetProgramType	103
7.17.2.5 ASX_PAD_GetProgramTypeString	103
7.17.2.6 ASX_PAD_GetRdsPI	104
7.17.2.7 ASX_PAD_GetTitle	104
7.18 Sample clock control functions	105
7.18.1 Detailed Description	105
7.18.2 Function Documentation	106
7.18.2.1 ASX_SampleClock_EnumerateClockSource	106
7.18.2.2 ASX_SampleClock_EnumerateLocalRate	106
7.18.2.3 ASX_SampleClock_EnumerateSampleRate	106
7.18.2.4 ASX_SampleClock_GetAutoSource	107
7.18.2.5 ASX_SampleClock_GetClockSource	107
7.18.2.6 ASX_SampleClock_GetLocalRate	107
7.18.2.7 ASX_SampleClock_GetLocalRateLock	108
7.18.2.8 ASX_SampleClock_GetSampleRate	108
7.18.2.9 ASX_SampleClock_SetAutoSource	108
7.18.2.10 ASX_SampleClock_SetClockSource	109
7.18.2.11 ASX_SampleClock_SetLocalRate	109
7.18.2.12 ASX_SampleClock_SetLocalRateLock	109
7.18.2.13 ASX_SampleClock_SetSampleRate	110
7.19 AESEBU receiver control functions	110
7.19.1 Detailed Description	110
7.19.2 Function Documentation	110
7.19.2.1 ASX_AESEBUReceiver_EnumerateFormat	110

7.19.2.2	ASX_AESEBUReceiver_GetErrorStatus	111
7.19.2.3	ASX_AESEBUReceiver_GetFormat	111
7.19.2.4	ASX_AESEBUReceiver_GetSampleRate	112
7.19.2.5	ASX_AESEBUReceiver_SetFormat	112
7.20	AESEBU transmitter control functions	112
7.20.1	Detailed Description	113
7.20.2	Function Documentation	113
7.20.2.1	ASX_AESEBUTransmitter_EnumerateFormat	113
7.20.2.2	ASX_AESEBUTransmitter_GetFormat	113
7.20.2.3	ASX_AESEBUTransmitter_SetFormat	113
7.21	GPIO control functions	114
7.21.1	Detailed Description	114
7.21.2	Function Documentation	114
7.21.2.1	ASX_GPIO_GetProperties	114
7.21.2.2	ASX_GPIO_InputGet	115
7.21.2.3	ASX_GPIO_OutputGet	115
7.21.2.4	ASX_GPIO_OutputSet	116
7.22	Vox control functions	116
7.22.1	Detailed Description	116
7.22.2	Function Documentation	117
7.22.2.1	ASX_Vox_GetLevel	117
7.22.2.2	ASX_Vox_GetRange	117
7.22.2.3	ASX_Vox_SetLevel	117
7.23	Generic control functions	118
7.23.1	Detailed Description	118
7.23.2	Function Documentation	118
7.23.2.1	ASX_GetGenericControlName	118
7.24	Microphone control functions	118
7.24.1	Detailed Description	119
7.24.2	Function Documentation	119
7.24.2.1	ASX_Mic_GetPhantomPower	119
7.24.2.2	ASX_Mic_SetPhantomPower	119
7.25	Parametric Equalizer control functions	119
7.25.1	Detailed Description	120
7.25.2	Function Documentation	120
7.25.2.1	ASX_EQ_GetBand	120
7.25.2.2	ASX_EQ_GetInfo	120
7.25.2.3	ASX_EQ_SetBand	121
7.25.2.4	ASX_EQ_SetState	121
7.26	Compander control functions	122
7.26.1	Detailed Description	123
7.26.2	Function Documentation	123
7.26.2.1	ASX_Compander_Get	123
7.26.2.2	ASX_Compander_GetAttackTimeConstant	123
7.26.2.3	ASX_Compander_GetDecayTimeConstant	124
7.26.2.4	ASX_Compander_GetEnable	124
7.26.2.5	ASX_Compander_GetMakeupGain	124
7.26.2.6	ASX_Compander_GetRatio	125
7.26.2.7	ASX_Compander_GetThreshold	125
7.26.2.8	ASX_Compander_Set	125

7.26.2.9	ASX_Compander_SetAttackTimeConstant	126
7.26.2.10	ASX_Compander_SetDecayTimeConstant	126
7.26.2.11	ASX_Compander_SetEnable	127
7.26.2.12	ASX_Compander_SetMakeupGain	127
7.26.2.13	ASX_Compander_SetRatio	127
7.26.2.14	ASX_Compander_SetThreshold	128
7.27	CobraNet control functions	128
7.27.1	Detailed Description	130
7.27.2	Function Documentation	130
7.27.2.1	ASX_Cobranet_EnumerateModes	130
7.27.2.2	ASX_Cobranet_GetConductorPriority	130
7.27.2.3	ASX_Cobranet_GetConductorStatus	130
7.27.2.4	ASX_Cobranet_GetDescription	131
7.27.2.5	ASX_Cobranet_GetErrorInfo	131
7.27.2.6	ASX_Cobranet_GetFirmwareRevision	132
7.27.2.7	ASX_Cobranet_GetIfStatus	132
7.27.2.8	ASX_Cobranet_GetIPAddress	132
7.27.2.9	ASX_Cobranet_GetLatencyAndSampleRate	133
7.27.2.10	ASX_Cobranet_GetLocation	133
7.27.2.11	ASX_Cobranet_GetMACAddress	133
7.27.2.12	ASX_Cobranet_GetMode	134
7.27.2.13	ASX_Cobranet_GetName	134
7.27.2.14	ASX_Cobranet_GetPersistence	134
7.27.2.15	ASX_Cobranet_GetSerialConfig	135
7.27.2.16	ASX_Cobranet_GetSerialEnable	135
7.27.2.17	ASX_Cobranet_GetStaticIPAddress	135
7.27.2.18	ASX_Cobranet_SetConductorPriority	136
7.27.2.19	ASX_Cobranet_SetIPAddress	136
7.27.2.20	ASX_Cobranet_SetLatencyAndSampleRate	136
7.27.2.21	ASX_Cobranet_SetLocation	137
7.27.2.22	ASX_Cobranet_SetMode	137
7.27.2.23	ASX_Cobranet_SetName	137
7.27.2.24	ASX_Cobranet_SetPersistence	138
7.27.2.25	ASX_Cobranet_SetSerialConfig	138
7.27.2.26	ASX_Cobranet_SetSerialEnable	138
7.27.2.27	ASX_Cobranet_SetStaticIPAddress	139
7.28	Cobranet transmitter control functions	139
7.28.1	Detailed Description	140
7.28.2	Function Documentation	140
7.28.2.1	ASX_CobranetTx_GetBundle	140
7.28.2.2	ASX_CobranetTx_GetChannelCount	140
7.28.2.3	ASX_CobranetTx_GetChannelMap	141
7.28.2.4	ASX_CobranetTx_GetFormat	141
7.28.2.5	ASX_CobranetTx_GetStatus	142
7.28.2.6	ASX_CobranetTx_GetUnicastMode	142
7.28.2.7	ASX_CobranetTx_SetBundle	142
7.28.2.8	ASX_CobranetTx_SetChannelCount	143
7.28.2.9	ASX_CobranetTx_SetChannelMap	143
7.28.2.10	ASX_CobranetTx_SetFormat	143
7.28.2.11	ASX_CobranetTx_SetUnicastMode	144

7.29	Cobranet receiver control functions	144
7.29.1	Detailed Description	145
7.29.2	Function Documentation	145
7.29.2.1	ASX_CobranetRx_GetBundle	145
7.29.2.2	ASX_CobranetRx_GetChannelMap	146
7.29.2.3	ASX_CobranetRx_GetMinimumDelay	146
7.29.2.4	ASX_CobranetRx_GetSourceMAC	146
7.29.2.5	ASX_CobranetRx_GetStatus	147
7.29.2.6	ASX_CobranetRx_SetBundle	148
7.29.2.7	ASX_CobranetRx_SetChannelMap	149
7.29.2.8	ASX_CobranetRx_SetMinimumDelay	149
7.29.2.9	ASX_CobranetRx_SetSourceMAC	149
7.30	Tone detector control functions	150
7.30.1	Detailed Description	150
7.30.2	Function Documentation	151
7.30.2.1	ASX_ToneDetector_GetEnable	151
7.30.2.2	ASX_ToneDetector_GetEventEnable	151
7.30.2.3	ASX_ToneDetector_GetFrequency	151
7.30.2.4	ASX_ToneDetector_GetState	151
7.30.2.5	ASX_ToneDetector_GetThreshold	152
7.30.2.6	ASX_ToneDetector_SetEnable	152
7.30.2.7	ASX_ToneDetector_SetEventEnable	152
7.30.2.8	ASX_ToneDetector_SetThreshold	153
7.31	Silence detector control functions	153
7.31.1	Detailed Description	153
7.31.2	Function Documentation	154
7.31.2.1	ASX_SilenceDetector_GetDelay	154
7.31.2.2	ASX_SilenceDetector_GetEnable	154
7.31.2.3	ASX_SilenceDetector_GetEventEnable	154
7.31.2.4	ASX_SilenceDetector_GetState	155
7.31.2.5	ASX_SilenceDetector_GetThreshold	155
7.31.2.6	ASX_SilenceDetector_SetDelay	155
7.31.2.7	ASX_SilenceDetector_SetEnable	155
7.31.2.8	ASX_SilenceDetector_SetEventEnable	156
7.31.2.9	ASX_SilenceDetector_SetThreshold	156
7.32	Block functions.	156
7.32.1	Detailed Description	157
7.32.2	Function Documentation	157
7.32.2.1	ASX_Block_GetInfo	157
7.32.2.2	ASX_Block_Parameter_Get	158
7.32.2.3	ASX_Block_Parameter_GetElementCount	158
7.32.2.4	ASX_Block_Parameter_GetEnumName	158
7.32.2.5	ASX_Block_Parameter_GetFlags	159
7.32.2.6	ASX_Block_Parameter_GetName	159
7.32.2.7	ASX_Block_Parameter_GetRange	160
7.32.2.8	ASX_Block_Parameter_GetType	160
7.32.2.9	ASX_Block_Parameter_GetUnits	160
7.32.2.10	ASX_Block_Parameter_Set	161

8.1	asxCobranetIpAutoassignParameters Struct Reference	163
8.1.1	Field Documentation	163
8.1.1.1	addr_end	163
8.1.1.2	addr_start	163
8.1.1.3	autoassign	163
8.2	asxParameterRangeInfo Struct Reference	163
8.2.1	Field Documentation	165
8.2.1.1	count	165
8.2.1.2	enumerated	165
8.2.1.3	enumerated_float	165
8.2.1.4	enumerated_integer	165
8.2.1.5	enums	165
8.2.1.6	floating	165
8.2.1.7	fmax	165
8.2.1.8	fmin	165
8.2.1.9	fstep	165
8.2.1.10	integer	165
8.2.1.11	max	165
8.2.1.12	max_len	165
8.2.1.13	min	165
8.2.1.14	step	165
8.2.1.15	string	165
8.2.1.16	type	165
8.2.1.17	u	165
8.2.1.18	value	165
8.2.1.19	value	165
8.3	asxParameterRangeInfo_NamedEnumerated Struct Reference	166
8.3.1	Field Documentation	166
8.3.1.1	name	166
8.3.1.2	value	166
8.4	asxParameterValue Struct Reference	166
8.4.1	Field Documentation	166
8.4.1.1	eType	166
8.4.1.2	size	166
8.4.1.3	uItems	166
8.4.1.4	value	166
9	File Documentation	167
9.1	asx.h File Reference	167
9.1.1	Define Documentation	202
9.1.1.1	_RPT0	202
9.1.1.2	_RPT1	202
9.1.1.3	ARRAY_SIZE	202
9.1.1.4	ASX32_API	202
9.1.1.5	ASX_LONG_STRING	202
9.1.1.6	ASX_LONGLONG_STRING	202
9.1.1.7	ASX_SHORT_STRING	202
9.1.2	Typedef Documentation	202
9.1.2.1	ASX_ERROR	202
9.1.2.2	ASX_ERROR_CALLBACK	203

9.1.2.3	ASX_HANDLE	203
9.1.2.4	ASX_NODE	203
9.1.2.5	ASX_PLAYER_CALLBACK	203
9.1.2.6	ASX_TIME	203
9.1.3	Enumeration Type Documentation	203
9.1.3.1	asxADAPTER_PROPERTY	203
9.1.3.2	asxADAPTERMODE	204
9.1.3.3	asxADPROPENUM_MODE	204
9.1.3.4	asxADPROPENUM_SSX2	204
9.1.3.5	asxAESEBU_FORMAT	205
9.1.3.6	asxAESEBU_STATUS	205
9.1.3.7	asxAUDIO_FORMAT	205
9.1.3.8	asxCHANNELMODE	206
9.1.3.9	asxCOBANET_IFSTATUS	206
9.1.3.10	asxCOBANET_LATENCY	206
9.1.3.11	asxCOBANET_MODE	207
9.1.3.12	asxCOMPANDER_INDEX	207
9.1.3.13	asxCONTROL	207
9.1.3.14	asxEQBANDTYPE	208
9.1.3.15	asxERROR	209
9.1.3.16	asxFILE_FORMAT	212
9.1.3.17	asxFILE_MODE	212
9.1.3.18	asxHANDLE_TYPE	212
9.1.3.19	asxMETER_TYPE	212
9.1.3.20	asxMSG_LOGGING	213
9.1.3.21	asxNODE	213
9.1.3.22	asxPLAYER_FLAGS	214
9.1.3.23	asxPLAYER_STATE	215
9.1.3.24	asxRECORD_MODE	215
9.1.3.25	asxRECORDER_STATE	215
9.1.3.26	asxSAMPLE_CLOCK_SOURCE	216
9.1.3.27	asxSAMPLE_RATE	218
9.1.3.28	asxTIMESCALE	218
9.1.3.29	asxTUNER_RDS_TYPE	219
9.1.3.30	asxTUNER_STATUS	219
9.1.3.31	asxTUNERBAND	219
9.1.3.32	asxTUNERDEEMPHASIS	220
9.1.3.33	asxTUNERHDBLEND	220
9.1.3.34	asxTUNERMODE	220
9.1.3.35	asxTUNERPROGRAM	220
9.1.3.36	asxUCONTROL_PFLAGS	221
9.1.3.37	asxUCONTROL_PTYPE	221
9.1.3.38	asxUCONTROL_RTYPE	222
9.1.3.39	asxVOLUME_AUTOFADE	222
9.2	asxstring.h File Reference	222
9.2.1	Define Documentation	223
9.2.1.1	ASX32_API	223
9.2.2	Function Documentation	223
9.2.2.1	ASXSTRING_EnumToString	223
9.2.2.2	ASXSTRING_StringToEnum	223

9.3	pcxport.txt File Reference	223
10	Example Documentation	225
10.1	adapter/main.c	225
10.2	cobranet/main.c	227
10.3	csharp_asx_player/Form1.cs	233
10.4	dual_mono_play/main.c	234
10.5	dual_mono_record/main.c	237
10.6	mixer/main.c	241
10.7	mux/main.c	246
10.8	play/main.c	248
10.9	playlist/main.c	251
10.10	record/main.c	256
10.11	system/main.c	259
10.12	tuner/main.c	260
10.13	volume/main.c	264

Chapter 1

Main Page

1.1 ASX Reference Manual

Table of contents

- [Introduction](#)
- [Installation and Usage](#)
- [Coding Conventions](#)
- [Compiling applications that use the ASX interface](#)
- [Functions](#)
- [Example code](#)
- [Porting Applications from PCXTools](#)
- [Deprecated](#)

1.2 Introduction

ASX is an audio API designed to work on both Windows and Linux operating systems. It is "high level", in that unlike say the Windows WAVE or Direct Sound APIs, which are "buffer based", it deals with audio files.

At a lower level, ASX supports the AudioScience HPI API (Windows and Linux) as well as WAVE and DirectSound (Windows) and ALSA (Linux).

The API is a C function library. This means it can be used from many languages including C, C++, C#, Java, Visual Basic and Delphi.

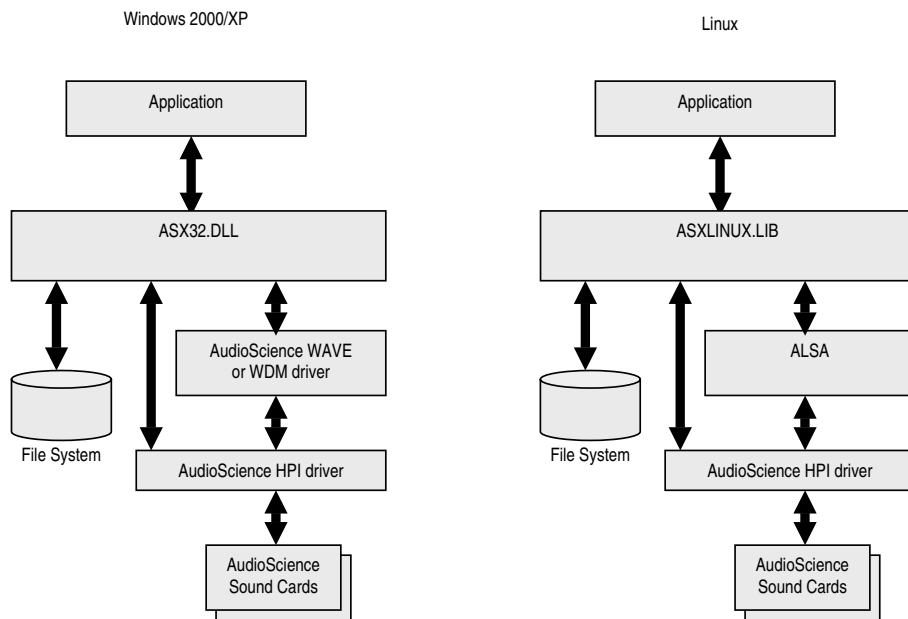


Figure 1.1: ASX interface under Windows and Linux.

1.3 Installation and Usage

1.3.1 Windows XP and Windows 7

1. Make sure you have an AudioScience WAVE, WDM or Combo driver installed and working with at least one audio adapter. The driver install will install both ASX32.DLL which implements the ASX interface documented here and ASIC-TRL.EXE which is a Win32 application that uses the ASX interface to control ASI adapters.
2. Obtain the ASX SDK executable file from the applications section of the AudioScience web site (<<http://www.audioscience.com/>>). The file is named ASX_SDK_WIN_XXXXX.EXE, where XXXXX is the version number.
3. Run the ASX_SDK_WIN_XXXXX.EXE application to install [asx.h](#), code examples and asx32.lib.

1.3.2 Linux

ToDo

1.4 Compiling applications that use the ASX interface

The various interface files required to interface with `asx32.dll` are typically installed in `c:\Program Files\AudioScience\ASX\lib` directory.

Applications can make use of `asx.h` and `asxstring.h` header files and should link against `asx.32lib`.

1.5 Debugging ASX calls under Windows

ASX ships with the capability of outputting debug information to a Third Party debug viewer.

DbgView should be used to display the debug messages. It is available from Microsoft and the simplest method of locating it is to just Google "DbgView". Download and install this on the PC you wish to test on.

The application being debugged should call `ASX_System_SetMessageLogging` to turn on error logging. Setting the error level to `asxMSG_LOGGING_DEBUG` will record all ASX calls and output other information as well.

1.6 ASX Object Model

In ASX, the underlying audio hardware is grouped into "adapters". An adapter is typically a soundcard (i.e. AudioScience ASI6114).

Each adapter has a "mixer". The mixer contains source and destination "nodes". A node represents a point at which audio comes in (source) or leaves (destination) the mixer.

Source nodes include:

- playback streams from the computer (usually from a file).
- physical audio inputs like an analog line input or AES/EBU input.

Destination nodes include:

- record streams to the computer (usually to a file).
- physical audio outputs like an analog line output or AES/EBU output.

Nodes are attached to "controls" which contain functionality to process the audio streams passing through them. Examples of controls include:

- Player control to play audio from a file on the computer.
- Recorder control to record audio to a file on the computer.
- Meter control to observe the audio signal's peak and RMS values.

- Volume control to alter the audio level.
- Channel Mode control to swap left and right channels on a stream.
- Level control to set the physical input and output levels on an analog node.
- AES/EBU control, which allows access to the channel status and user data.

The following diagram shows how ASX models a simple sound card:

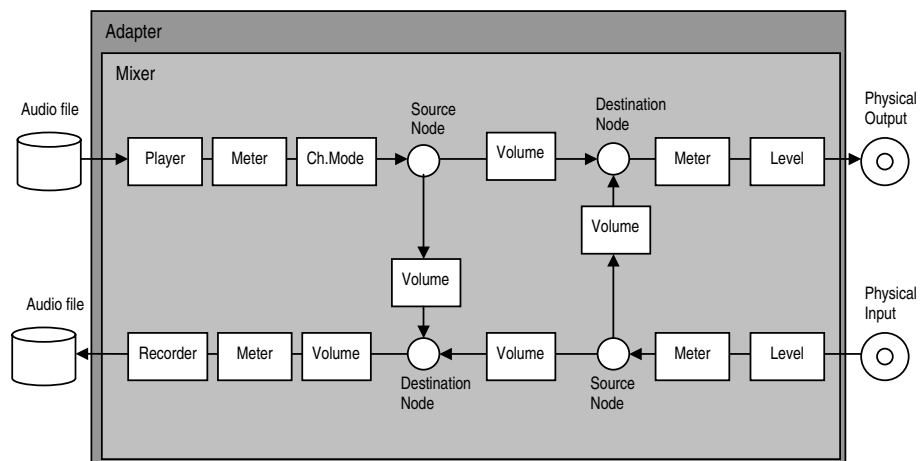


Figure 1.2: ASX controls within a sound card.

1.7 Coding Conventions

Method names

- 1st letters are ASX_ int uppercase, i.e [ASX_Adapter_GetMixer\(\)](#)

Variable names use Hungarian notation, i.e.

- int - prefix with "n", i.e int nIndex
- long - prefix with "l", i.e long lIndex
- unsigned long - prefix with "dw", i.e unsigned long dwSampleRate;
- float - prefix with "f", i.e float fNumber;
- pointer - prefix with "p", i.e int* pIndex
- reference - prefix with "r", i.e int& rIndex
- string (char array) - prefix with "sz", i.e char szString[40];
- global var - prefix with "g", i.e int gIndex
- static var - prefix with "s", i.e int sIndex
- member var - prefix with "m", i.e int mIndex
- enum - prefix with "asx", then all capitals with "_" separators, i.e. asxERROR_INDEX_OUT_OF_RANGE

Return conventions All functions return ASX_ERROR which equals 0 on success and non-zero on error.

Chapter 2

Porting Applications from PCXTools

The following is a guide to converting an application that uses Digigram's PCXTools API to use the ASX API.

2.1 Global

2.1.1 TOOLRegister(),ToolUnregister()

Call ASX_CreateSystem() and ASX_DeleteSystem().

2.1.2 TOOLSetOEMAuthorizationDll

No corresponding ASX call

2.1.3 TOOLGetLastError, TOOLGetErrorString

Call ASX_GetLastError() and ASX_GetLastErrorString()

2.2 Configuration

2.2.1 TOOLGetVersions

2.2.2 TOOLHowManyTotalOutputs

2.2.3 TOOLHowManyTotalInputs

2.2.4 TOOLGetBoardInfos

2.2.5 TOOLGetBoardName

2.3 Pipe

2.3.1 TOOLAllocatePipe

2.3.2 TOOLAllocatePipeEx

2.3.3 TOOLReleasePipe

2.3.4 TOOLPipeStart

2.3.5 TOOLPipeStop

2.3.6 TOOLPipeSetLevel

2.3.7 TOOLPipeSetMonitoringMute

2.3.8 TOOLPipeSetClock

2.3.9 TOOLPipeDefineDigitalInput

2.3.10 TOOLPipeSetStretch

2.3.11 TOOLPipeGetVuMeter

2.3.12 TOOLPipeGetClock

2.3.13 TOOLPipeGetPosition

2.4 Stream

TOOLPlayFile

TOOLPipeScrubFile
TOOLPlayFileWithLevels
TOOLRecordFile
TOOLStreamStop
TOOLStreamSetDigitalLevel
TOOLStreamSetPanLevel
TOOLStreamGoToLevel
TOOLStreamSetRecordFormat
TOOLStreamSetMpegEqualization
TOOLStreamGetPosition
TOOLStreamGetStatus

2.5 Offline

TOOLOfflineGetProgress
TOOLOfflineAbortOperation
TOOLOfflineReserveChannel
TOOLOfflineSetAncillaryData
TOOLOfflineLinkPipes

2.6 Misc

TOOLGetOffsetInputLevel
TOOLGetOffsetOutputLevel
TOOLConvertEx
TOOLConvertPart
TOOLConvertSetLevel
TOOLStretchEx
TOOLFileInfo
TOOLFileGetNormalizeInfo
TOOLGetBoardExternalClock
TOOLGetDigitalInputExternalClock

Chapter 3

Deprecated List

Global [ASX_Cobranet_EnumerateModes](#)(ASX_HANDLE hCobranet, const int nIndex, enum asxCOBANET_MODE *peMode)

This function has been removed (it is stubbed out).

Global [ASX_Cobranet_GetMode](#)(ASX_HANDLE hCobranet, enum asxCOBANET_MODE *peMode)

This function has been removed (it is stubbed out).

Global [ASX_Cobranet_SetMode](#)(ASX_HANDLE hCobranet, const enum asxCOBANET_MODE eMode)

This function has been removed (it is stubbed out).

Global [ASX_Compander_Get](#)(ASX_HANDLE hCompander, unsigned short *pwAttack, unsigned short *pwDecay, signed short *pwGain, signed short *pwRelease)

This function has been broekn up in to individual Get() functions. Gets the parameters for the compander.

Global [ASX_Compander_Set](#)(ASX_HANDLE hCompander, const unsigned short wAttack, const unsigned short wDecay, const signed short wGain, const signed short wRelease)

This function has been broekn up in to individual Set() functions. Sets the parameters for the compander.

Global [ASX_Mixer_GetNodeIndex](#)(ASX_HANDLE hNode, int *pnIndex) This function has been superseded by [ASX_Node_GetIndex\(\)](#)

Global [ASX_Mixer_GetNodeType](#)(ASX_HANDLE hNode, enum asxNODE *peType)

This function has been superseded by [ASX_Node_GetType\(\)](#)

Global [ASX_SampleClock_EnumerateSampleRate](#)(ASX_HANDLE hSampleClock, const int nIndex, enum asxSAMPLE_RATE *peRate)

This function has been superseded by [ASX_SampleClock_EnumerateLocalRate\(\)](#)

Global [ASX_SampleClock_SetSampleRate](#)(ASX_HANDLE hSampleClock, const int nSampleRate)

This function has been superseded by [ASX_SampleClock_SetLocalRate\(\)](#)

Global [ASX_Tuner_GetHdRadioDspVersion](#)(ASX_HANDLE hTuner, char *szSdkVersion, const int nStr)

This function has been superseded by [ASX_Tuner_GetFirmwareVersion\(\)](#)

Global [ASX_Tuner_GetHdRadioSdkVersion](#)(ASX_HANDLE hTuner, char *szSdkVersion, const int nStr)

This function has been superseded by [ASX_Tuner_GetFirmwareVersion\(\)](#)

Global [ASX_Tuner_GetHdRadioSignalQuality](#)(ASX_HANDLE hTuner, int *pnSignalQuality)

This function has been superseded by [ASX_Tuner_GetDigitalSignalQuality\(\)](#)

Chapter 4

Module Index

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

asxCobranetIpAutoassignParameters	163
asxParameterRangeInfo	163
asxParameterRangeInfo_NamedEnumerated	166
asxParameterValue	166

Chapter 6

File Index

Chapter 7

Module Documentation

7.1 SubSystem types

The types of audio subsystems that can operate using the ASX interface.

Defines

- #define [ASX_SYSTEM_TYPE_HPI](#) 0
Use this to select ASI's HPI interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_WAVE](#) 1
Use this to select Microsoft's WAVE interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_ALSA](#) 2
Use this to select the Linux ALSA interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_DIRECTX](#) 3
Use this to select Microsoft's DirectX interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_PORTAUDIO](#) 4
Use this to select the PortAudio interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_PCXTOOLS](#) 5
Use this to select Digigram's PCX interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_SNMP](#) 6
Use this to select Cobranet SNMP interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_HPIUDP](#) 7
Use this to select ASI's HPI interface over UDP when calling [ASX_System_Create\(\)](#). Also supports HPI.

- `#define ASX_SYSTEM_TYPE_DUMMY 8`
Dummy backend.
- `#define ASX_SYSTEM_TYPE_ANY 9`
Wild card - any subsystem (reserved).
- `#define ASX_SYSTEM_TYPE_ASIO 10`
Use this to select Stienberg's ASIO interface when calling [ASX_System_Create\(\)](#).
- `#define ASX_SYSTEM_TYPE_AVB_1722_1 11`
Use this to select the IEEE 1722.1 system for controlling AVB devices when calling [ASX_System_Create\(\)](#).
- `#define ASX_SYSTEM_TYPE_COUNT 12`
Indicates the number of subsystems defined.

7.1.1 Detailed Description

The types of audio subsystems that can operate using the ASX interface.

7.1.2 Define Documentation

7.1.2.1 `#define ASX_SYSTEM_TYPE_ALSA 2`

Use this to select the Linux ALSA interface when calling [ASX_System_Create\(\)](#).

7.1.2.2 `#define ASX_SYSTEM_TYPE_ANY 9`

Wild card - any subsystem (reserved).

7.1.2.3 `#define ASX_SYSTEM_TYPE_ASIO 10`

Use this to select Stienberg's ASIO interface when calling [ASX_System_Create\(\)](#).

7.1.2.4 `#define ASX_SYSTEM_TYPE_AVB_1722_1 11`

Use this to select the IEEE 1722.1 system for controlling AVB devices when calling [ASX_System_Create\(\)](#).

7.1.2.5 `#define ASX_SYSTEM_TYPE_COUNT 12`

Indicates the number of subsystems defined.

7.1.2.6 #define ASX_SYSTEM_TYPE_DIRECTX 3

Use this to select Microsoft's DirectX interface when calling [ASX_System_Create\(\)](#).

7.1.2.7 #define ASX_SYSTEM_TYPE_DUMMY 8

Dummy backend.

7.1.2.8 #define ASX_SYSTEM_TYPE_HPI 0

Use this to select ASI's HPI interface when calling [ASX_System_Create\(\)](#).

Examples:

[adapter/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [record/main.c](#), [system/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.1.2.9 #define ASX_SYSTEM_TYPE_HPIUDP 7

Use this to select ASI's HPI interface over UDP when calling [ASX_System_Create\(\)](#). Also supports HPI.

Examples:

[cobranet/main.c](#).

7.1.2.10 #define ASX_SYSTEM_TYPE_PCXTOOLS 5

Use this to select Digigram's PCX interface when calling [ASX_System_Create\(\)](#).

7.1.2.11 #define ASX_SYSTEM_TYPE_PORTAUDIO 4

Use this to select the PortAudio interface when calling [ASX_System_Create\(\)](#).

7.1.2.12 #define ASX_SYSTEM_TYPE_SNMP 6

Use this to select Cobranet SNMP interface when calling [ASX_System_Create\(\)](#).

Examples:

[cobranet/main.c](#).

7.1.2.13 `#define ASX_SYSTEM_TYPE_WAVE 1`

Use this to select Microsoft's WAVE interface when calling [ASX_System_Create\(\)](#).

7.2 System functions

Functions

- [ASX32_API int ASX_System_SupportsSubSystem](#) (const int asxSystemType)
Query ASX library for subsystem support.
- [ASX_System_Create](#)
Create a complete ASX system.
- [ASX_System_CreateSubSystem](#)
Creates an ASX sub system and adds it to the existing system, if any.
- [ASX_System_SetHostNetworkInterface](#)
Set the interface ASX should use when communicating with network devices.
- [ASX_System_SetAvdeccInterface](#)
Set the interface ASX should use when initializing the AVDECC controller.
- [ASX_System_Delete](#)
Delete a complete ASX system.
- [ASX_System_RegisterErrorCallback](#)
Register a callback function that should be called when an error is detected.
- [ASX_System_GetName](#)
Gets the name of the ASX system.
- [ASX_System_GetVersion](#)
Get ASX system version information.
- [ASX_System_GetAdapterCount](#)
Get the number of adapters.
- [ASX_System_GetAdapter](#)
Get a handle to a specific adapter.
- [ASX_System_SetMessageLogging](#)
Set the message logging level for ASX.
- [ASX_System_GetMessageLogging](#)

Get the message logging level for ASX.

- [ASX_System_SetCobranetAutoassignParms](#)

Set the IP address range that will be used for assigning IP addresses to cobranet devices.

- [ASX_System_GetCobranetAutoassignParms](#)

Get the IP address range that will be used for assigning IP addresses to cobranet devices.

7.2.1 Function Documentation

7.2.1.1 ASX32_API ASX_ERROR ASX_System.Create (const int *asxSystemType*, ASX_HANDLE * *phSystem*)

Create a complete ASX system.

This function creates a complete ASX interface of the type specified by *asxSystemType*. If more than one system type is needed use [ASX_System_CreateSubSystem\(\)](#) instead.

Parameters

<i>asxSystemType</i>	The ASX system type to open. One of SubSystem types defines above.
<i>phSystem</i>	Pointer to the returned system handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [record/main.c](#), [system/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.2.1.2 ASX32_API ASX_ERROR ASX_System.CreateSubSystem (const int *asxSystemType*, ASX_HANDLE * *pio.hSystem*)

Creates an ASX sub system and adds it to the existing system, if any.

Use this function when creating more than one subsystem.

Note

Starting with driver version 4.18.00 when this function is called to create the HPI UDP subsystem it will only add one found adapter before returning; call [ASX_System_GetAdapterCount\(\)](#) to add more.

Parameters

<i>asxSystem-Type</i>	The ASX system type to open. One of SubSystem types defines above.
<i>pio_hSystem</i>	Pointer to the system handle. Should be a pointer to NULL for the first call.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.2.1.3 ASX32_API ASX_ERROR ASX_System_Delete (ASX_HANDLE hSystem)

Delete a complete ASX system.

The delete function should be called using a previously opened ASX system handle prior to closing an application. Note that when an ASI2416 CobraNet device is in use a call to [ASX_System_Delete\(\)](#) triggers a save to ASI2416 flash of any control parameters that might have changed. If this fails for any reason an [asxERROR_MIXER_SAVECONTROLSTATE](#) error is returned.

Parameters

<i>hSystem</i>	The asx system handle.
----------------	------------------------

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#), [cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [record/main.c](#), [system/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.2.1.4 ASX32_API ASX_ERROR ASX_System_GetAdapter (ASX_HANDLE hSystem, const int nAdapter, ASX_HANDLE * p_hAdapter)

Get a handle to a specific adapter.

This function returns a handle to an adapter object that can then be used to access functionality of the adapter.

Parameters

<i>hSystem</i>	A handle to an ASX system object.
<i>nAdapter</i>	The index of the adapter.
<i>p_hAdapter</i>	The returned adapter handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#), [cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#),
[mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [record/main.c](#), [tuner/main.c](#),
and [volume/main.c](#).

7.2.1.5 ASX32_API ASX_ERROR ASX_System.GetAdapterCount (ASX_HANDLE *hSystem*, int * *pnCount*)

Get the number of adapters.

This function returns the number of recognized sound cards installed in the computer.

Note

Starting with driver version 4.18.00 the HPI UDP subsystem will only add one found network adapter at a time before returning from this call; call it repeatedly (until the count stops incrementing) to find all networked adapters and periodically thereafter to detect new network adapter connections

Parameters

<i>hSystem</i>	A handle to an ASX system object.
<i>pnCount</i>	The returned number of adapters.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#), [cobranet/main.c](#), and [system/main.c](#).

7.2.1.6 ASX32_API ASX_ERROR ASX_System.GetCobranetAutoassignParms (struct [asxCobranetIpAutoassignParameters](#) * *pCAP*)

Get the IP address range that will be used for assigning IP addresses to cobranet devices.

Parameters

<i>pCAP</i>	Pointer to the asxCobranetIpAutoassignParameters structure that receives the IP autoassign parameters.
-------------	--

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.2.1.7 ASX32_API ASX_ERROR ASX_System_GetMessageLogging (ASX_HANDLE *hSystem*, enum asxMSG_LOGGING * *eLog*)

Get the message logging level for ASX.

Parameters

<i>hSystem</i>	A handle to an ASX system object.
<i>eLog</i>	The error logging level. see asxMSG_LOGGING for options

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.2.1.8 ASX32_API ASX_ERROR ASX_System_GetName (ASX_HANDLE *hSystem*, char * *pszName*, const int *nStringLength*, int * *pnRequiredLength*)

Gets the name of the ASX system.

This function returns the name of the audio substem currently being used underneath ASX, for example, "HPI" would be returned if ASX was being run using the Audio-Science HPI driver.

Parameters

<i>hSystem</i>	The asx system handle.
<i>pszName</i>	The string to use to copy the returned adapter name to. Typical return values are: "HPI" "Wave" "DirectX" "PortAudio" "ALSA"
<i>nStringLength</i>	The length of the string <i>szString</i> that was passed in.
<i>pnRequiredLength</i>	The minimum required length in bytes of <i>szString</i> .

Note

This function can be called with *szString*=0 and *nStringLength*=0 to retrieve the string size required.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[system/main.c](#).

7.2.1.9 ASX32_API ASX_ERROR ASX_System_GetVersion (
ASX_HANDLE hSystem, char * pszSystemVersion,
const int nSystemVersionLength, int * pnRequiredSystemVersionLength,
char * pszSubSystemVersion, const int nSubSystemVersionLength, int *
pnRequiredSubSystemVersionLength)

Get ASX system version information.

This function returns version information in two strings.

Parameters

<i>hSystem</i>	A handle to an ASX system object.
<i>pszSystemVersion</i>	The ASX version returned as a string.
<i>nSystemVersionLength</i>	The length of pszSystemVersion in bytes.
<i>pnRequiredSystemVersionLength</i>	The minimum required length of pszSystemVersion in bytes.
<i>pszSubSystemVersion</i>	The ASX subsystem version returned as a string. This is the version number of the HPI, Wave or ALSA (etc.) driver.
<i>nSubSystemVersionLength</i>	The length in bytes of pszSubSystemVersion.
<i>pnRequiredSubSystemVersionLength</i>	The minimum required length in bytes of pszSubSystemVersion.

Note

This function can be called with string pointers set to zero to determine the size of strings to allocate.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[system/main.c](#).

7.2.1.10 ASX32_API ASX_ERROR ASX_System.RegisterErrorCallback (
ASX_HANDLE *hSystem*, ASX_ERROR_CALLBACK *
***pCallback*, void * *pUser1*, void * *pUser2*)**

Register a callback function that should be called when an error is detected.

Note that using this function is optional. [ASX_Error_GetLast\(\)](#) can be used after each call, or the error return value of each call may be checked. The callback function itself should use [ASX_Error_GetLast\(\)](#) to figure out what error was actually generated.

Parameters

<i>hSystem</i>	A handle to an ASX system object.
<i>pCallback</i>	A pointer to a callback of type ASX_ERROR_CALLBACK.
<i>pUser1</i>	A user defined pointer that is passed back when an error occurs.
<i>pUser2</i>	A user defined pointer that is passed back when an error occurs.

Note

An error of type [asxERROR_INVALID_CONTROL_ATTRIBUTE](#) returned by control operations that attempt to access functionality not supported by the control will not cause a callback to *pCallback* routine.

7.2.1.11 ASX32_API ASX_ERROR ASX_System.SetAvdeccInterface (
const char * *szInterface*)

Set the interface ASX should use when initializing the AVDECC controller.

This function should be called before [ASX_System_Create\(\)](#).

Parameters

<i>szInterface</i>	The AVDECC interface string to use.
--------------------	-------------------------------------

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.2.1.12 ASX32_API ASX_ERROR ASX_System.SetCobranetAutoassignParms (
const struct asxCobranetIpAutoassignParameters * *pCAP*)

Set the IP address range that will be used for assigning IP addresses to cobranet devices.

This function should be called before the ASX_SYSTEM_TYPE_SNMP is create with [ASX_System_Create\(\)](#) or [ASX_System_CreateSubSystem\(\)](#).

Parameters

<i>pCAP</i>	Pointer to the asxCobranetIpAutoassignParameters structure that contains the IP autoassign parameters.
-------------	--

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.2.1.13 ASX32_API ASX_ERROR ASX_System.SetHostNetworkInterface (
const char * szInterface)

Set the interface ASX should use when communicating with network devices.

This function should be called before [ASX_System_Create\(\)](#).

Parameters

<i>szInterface</i>	The network interface to use. For Windows this is the IP address of the form "192.168.1.13".
--------------------	--

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.2.1.14 ASX32_API ASX_ERROR ASX_System.SetMessageLogging (
ASX_HANDLE hSystem, const enum asxMSG_LOGGING
eLog)

Set the message logging level for ASX.

Parameters

<i>hSystem</i>	A handle to an ASX system object.
<i>eLog</i>	The error logging level to set.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#).

7.2.1.15 ASX32_API int ASX_System.SupportsSubSystem (
const int asxSystemType)

Query ASX library for subsystem support.

Indicate if this instance of ASX library supports the given subsystem type.

Parameters

<i>asxSystem-Type</i>	The ASX system type to query. One of SubSystem types defines above.
-----------------------	---

Returns

1 if asxSystemType type supported, 0 if asxSystemType is not supported.

7.3 Handle functions

The error functions operate on all ASX objects to collect, report and clear errors.

Functions

- ASX32_API enum [asxHANDLE_TYPE](#) [ASX_Handle_GetType](#) ([ASX_HANDLE](#) hHandle)

Get the handle type.

7.3.1 Detailed Description

The error functions operate on all ASX objects to collect, report and clear errors.

7.3.2 Function Documentation

7.3.2.1 ASX32_API enum [asxHANDLE_TYPE](#) [ASX_Handle_GetType](#) ([ASX_HANDLE](#) hHandle)

Get the handle type.

Parameters

<i>hHandle</i>	A handle to any type of ASX object.
----------------	-------------------------------------

Returns

Returns one of [asxHANDLE_TYPE](#).

7.4 Error functions

The error functions operate on all ASX objects to collect, report and clear errors.

Functions

- [ASX_Error_GetLast](#)

Get the last error.

- [ASX_Error_GetLastString](#)

Get the last error string information.

- [ASX_Error_Clear](#)

Clears the last error.

7.4.1 Detailed Description

The error functions operate on all ASX objects to collect, report and clear errors.

7.4.2 Function Documentation

7.4.2.1 ASX32_API ASX_ERROR ASX_Error_Clear (ASX_HANDLE hASXObject)

Clears the last error.

This function clears error information for the last error generated by hASXObject.

Parameters

<i>hASXObject</i>	The ASX object handle.
-------------------	------------------------

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.4.2.2 ASX32_API ASX_ERROR ASX_Error_GetLast (ASX_HANDLE hASXObject, ASX_ERROR * pnAsxErrorCode, int * pnAsxSubSystemErrorCode)

Get the last error.

This function returns error information for the last error generated by hASXObject.

Parameters

<i>hASXObject</i>	The ASX object handle that generated the error. This can be any ASX object.
<i>pnAsxError-Code</i>	The returned ASX error code. If this parameter is set to 0, it will be ignored.

<i>pnAsxSubSystemError</i> <i>Code</i>	The returned SubSystem error code. This code will be an HPI error if the HPI subsystem is being used, or a MMSYSTEM error if MMSYSTEM is being use. If this parameter is set to 0, it will be ignored.
---	--

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#), [cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [record/main.c](#), [system/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.4.2.3 ASX32_API ASX_ERROR ASX_Error_GetLastString (
ASX_HANDLE hASXObject, char * pszAsxErrorString,
const int nAsxErrorStringLength, int * pnRequiredAsxErrorStringLength, char *
pszAsxSubSystemErrorString, const int nAsxSubSystemErrorStringLength, int *
pnRequiredAsxSubSystemErrorStringLength)

Get the last error string information.

This function returns error information for the last error generated by hASXObject. Note that errors are automatically cleared the next time any operation is performed using hASXObject.

Parameters

<i>hASXObject</i>	The ASX object handle that generated the error. This can be any ASX object.
<i>pszAsxErrorString</i>	The returned ASX error string. If this parameter is set to 0, it will be ignored.
<i>nAsxErrorStringLength</i>	The length of pszAsxErrorString on bytes.
<i>pnRequiredAsxErrorStringLength</i>	The required length of pszAsxErrorString in bytes.
<i>pszAsxSubSystemErrorString</i>	The returned SubSystem error string. This string will describe an HPI error if the HPI subsystem is being used, or a MMSYSTEM error if MMSYSTEM is being use. If this parameter is set to 0, it will be ignored.
<i>nAsxSubSystemErrorStringLength</i>	The length of pszAsxSubSystemErrorString in bytes.
<i>pnRequiredAsxSubSystemErrorStringLength</i>	The required length of pszAsxSubSystemErrorString in bytes.

Note

This function can be called with string pointers set to zero to determine the size of strings to allocate.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#), [cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [record/main.c](#), [system/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.5 Adapter functions

The adapter functions are used to obtain adapter information and access the mixer.

Functions

- [ASX_Adapter_CheckSubSystems](#)
Returns the status of the various sub-systems that interface to the adapter.
- [ASX_Adapter_GetName](#)
Gets the name of the adapter.
- [ASX_Adapter_GetIndex](#)
Gets an adapter's index.
- [ASX_Adapter_GetAVDECCController](#)
Gets an adapter's AVDECC Controller object.
- [ASX_Adapter_GetAVDECCSystem](#)
This function returns the system object used by adapters of subsystem type 1722.1.
- [ASX_Adapter_GetSerialNumber](#)
Gets an adapter's serial number.
- [ASX_Adapter_GetHardwareRevision](#)
Gets an adapter's hardware revision.
- [ASX_Adapter_GetFirmwareRevision](#)
Gets an adapter's firmware revision.
- [ASX_Adapter_GetMacAddress](#)
Gets an adapter's ethernet MAC address.

- [ASX_Adapter_GetIpAddress](#)
Gets an adapter's network IP address.
- [ASX_Adapter_GetDspUtilization](#)
Gets an adapter's DSP utilization.
- [ASX_Adapter_GetMixer](#)
Gets a handle to an adapter's mixer.
- [ASX_Adapter_EnumerateMode](#)
Enumerate each adapter mode option.
- [ASX_Adapter_GetMode](#)
Get the current adapter mode.
- [ASX_Adapter_SetMode](#)
Set the current adapter mode.
- [ASX_Adapter_EnumerateProperty](#)
Enumerates adapter properties and settings.
- [ASX_Adapter_ReadProperty](#)
Read an adapter's property value.
- [ASX_Adapter_WriteProperty](#)
Write an adapter property value.
- [ASX_Adapter_WriteNvMem](#)
Write a byte to the non-volatile memory.
- [ASX_Adapter_ReadNvMem](#)
Read a byte from the non-volatile memory.
- [ASX_Adapter_GetNvMemSizeInBytes](#)
Get the number of bytes in the adapter's non-volatile memory.

7.5.1 Detailed Description

The adapter functions are used to obtain adapter information and access the mixer.

7.5.2 Function Documentation

7.5.2.1 ASX32_API ASX_ERROR ASX_Adapter_CheckSubSystems (
ASX_HANDLE *hAdapter*, unsigned int * *pnSubSystemMask*,
unsigned int * *pnSubSystemOkMask*)

Returns the status of the various sub-systems that interface to the adapter.

This function is primarily implemented to provide feedback on whether adapters (or subsystems) that a network interface are working correctly. The bit masks used in this function consist of 1<<ASX_SYSTEM_TYPE_XXXX.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pnSubSystemMask</i>	A bit mapped mask of the subsystems that interface to this adapter.
<i>pnSubSystemOkMask</i>	A bit mapped result indicating that a particular interface is ok.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.2 ASX32_API ASX_ERROR ASX_Adapter_EnumerateMode (
ASX_HANDLE *hAdapter*, const int *nIndex*, enum
asxADAPTERMODE * *peMode*, int * *pnCount*)

Enumerate each adapter mode option.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>nIndex</i>	The index of the adapter mode option to fetch.
<i>peMode</i>	The returned adapter mode option.
<i>pnCount</i>	The total number of available mode options. See asxADAPTERMODE for available options.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.3 ASX32_API ASX_ERROR ASX_Adapter_EnumerateProperty (
ASX_HANDLE *hAdapter*, const int *nIndex*, const enum
asxADPROPENUM_MODE *eMode*, const int *nSubIndex*, unsigned long *
***pdwSetting*)**

Enumerates adapter properties and settings.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>nIndex</i>	The property number.
<i>eMode</i>	Enumeration mode (See <code>asxADPROPENUM_MODE</code>).
<i>nSubIndex</i>	Subindex.
<i>pdwSetting</i>	Returned setting.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.4 ASX32_API ASX_ERROR ASX_Adapter.GetAVDECCController (ASX_HANDLE hAdapter, void ** controller_obj)

Gets an adapter's AVDECC Controller object.

This function returns the controller object used by adapters of subsystem type 1722.1

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>controller</i>	The returned controller object.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.5 ASX32_API ASX_ERROR ASX_Adapter.GetAVDECCSystem (ASX_HANDLE hAdapter, void ** sys)

This function returns the system object used by adapters of subsystem type 1722.1.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>controller</i>	The returned system object.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.6 ASX32_API ASX_ERROR ASX_Adapter.GetDspUtilization (ASX_HANDLE hAdapter, const int nDspIndex, int * pnDspUtilization)

Gets an adapter's DSP utilization.

This function returns the DSP percentage utilization of the audio adapter referenced by

hAdapter. The utilization can be used to check the running algorithms do not over tax the DSP.

Note

This function is currently only supported by the ASX_SYSTEM_TYPE_HPI interface.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>nDspIndex</i>	The DSP index. On adapters with more than one DSP, all DSPs can be accessed using this index field.
<i>pnDspUtilization</i>	The returned DSP utilization in percent.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#).

7.5.2.7 ASX32_API ASX_ERROR ASX_Adapter_GetFirmwareRevision (ASX_HANDLE *hAdapter*, char * *pszRevision*)

Gets an adapter's firmware revision.

This function returns the revision of the firmware running on an audio adapter referenced by *hAdapter*. An example of a revision string is "v1.25".

Note

This function is currently only supported by the ASX_SYSTEM_TYPE_HPI interface.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pszRevision</i>	A pointer to a char array of length ASX_SHORT_STRING to return the revision string.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.8 ASX32_API ASX_ERROR ASX_Adapter_GetHardwareRevision (ASX_HANDLE *hAdapter*, char * *pszRevision*)

Gets an adapter's hardware revision.

This function returns the revision of the audio adapter referenced by `hAdapter`. The revision indicates the hardware revision of the adapter. An example of a revision string is "A0". The first character is a letter (A-Z) indicating the major revision number and the second character is a digit (0-9) indicating the minor revision number.

Note

This function is currently only supported by the `ASX_SYSTEM_TYPE_HPI` interface.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pszRevision</i>	A pointer to a char array of length <code>ASX_SHORT_STRING</code> to return the revision string.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#).

7.5.2.9 ASX32_API ASX_ERROR ASX_Adapter_GetIndex (ASX_HANDLE hAdapter, int * pnIndex)

Gets an adapter's index.

This function returns the hardware index of the audio adapter referenced by `hAdapter`. All AudioScience adapters have unique indexes assigned by a hardware jumper/switch (sound cards) or programmed into non-volatile memory (network devices). Indexes in the range of 1..99 are used for sound cards or other bus based devices. Indexes in the range of 100..9999 are used for network devices. Indexes of 10000 and higher are used to auto-assign indexes to network devices that don't have an index programmed yet. Note that the hardware index is not the same as the ASX adapter index passed to [ASX_System_GetAdapter\(\)](#).

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pnIndex</i>	The returned index.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#), and [cobranet/main.c](#).

7.5.2.10 ASX32_API ASX_ERROR ASX_Adapter_GetIpAddress (ASX_HANDLE *hAdapter*, char * *pszIP*)

Gets an adapter's network IP address.

For network devices, this function returns the IP address of the Adapter. The IP address is returned as a string with the format XXX.XXX.XXX.XXX. If the adapter does not have an IP address (i.e if it was a sound card) then an error is returned.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pszIP</i>	A pointer to a char array of length ASX_SHORT_STRING to return the MAC address string.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.5.2.11 ASX32_API ASX_ERROR ASX_Adapter_GetMacAddress (ASX_HANDLE *hAdapter*, char * *pszMAC*)

Gets an adapter's ethernet MAC address.

For network devices, this function returns the ethernet MAC address of the Adapter. The MAC address is returned as a string representing the 12 hex digits, with the format XXXX.XXXX.XXXX. If the adapter does not have a MAC address (i.e if it was a sound card) then an error is returned.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pszMAC</i>	A pointer to a char array of length ASX_SHORT_STRING to return the MAC address string.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[mixer/main.c](#).

7.5.2.12 ASX32_API ASX_ERROR ASX_Adapter_GetMixer (ASX_HANDLE *hAdapter*, ASX_HANDLE * *p.hMixer*)

Gets a handle to an adapter's mixer.

This function returns a handle to a mixer object that can then be used to access mixer nodes and controls.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>p_hMixer</i>	The returned mixer handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [record/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.5.2.13 ASX32_API ASX_ERROR ASX_Adapter_GetMode (ASX_HANDLE hAdapter, enum asxADAPTERMODE * peMode)

Get the current adapter mode.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>peMode</i>	The returned adapter mode. See asxADAPTERMODE for different mode options.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.14 ASX32_API ASX_ERROR ASX_Adapter_GetName (ASX_HANDLE hAdapter, char * pszName, const int nStringLength, int * RequiredLength)

Gets the name of the adapter.

This function returns the name of the audio adapter referenced by hAdapter. An example return string would be "ASI6114" for and AudioScience ASI6114 adapter.

Note that some adapters have plug in modules and will return addition module code characters following the adapter name. For example a ASI8920 with two modules would return "ASI8920-1100", indicating that module positions 1 and 2 are populated with modules of type "1". The device datasheet, in this case the ASI8900, should be consulted to translate the module code to a module type.

```
// an example of the "compact" calling method would be
char szName[ASX_SHORT_STRING];
```

```
ASX_Adapter_GetName(hAdapter, szName, ASX_SHORT_STRING, NULL);
```

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pszName</i>	The string to use to copy the returned adapter name to.
<i>nStringLength</i>	The length of szString in bytes.
<i>RequiredLength</i>	The minimum required length of szString in bytes. This can be set to NULL if the calling passes in a string of length ASX_SHORT_STRING.

Note

This function can be called with szString=0 and nStringLength=0 to retrieve the required string size in bytes.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#), [cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.5.2.15 ASX32_API ASX_ERROR ASX_Adapter_GetNvMemSizeInBytes (ASX_HANDLE hAdapter, int * pnCount)

Get the number of bytes in the adapter's non-volatile memory.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pnCount</i>	A pointer to the returned total number of bytes.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.16 ASX32_API ASX_ERROR ASX_Adapter_GetSerialNumber (ASX_HANDLE hAdapter, unsigned long * pdwSerialNumber)

Gets an adapter's serial number.

This function returns the serial number of the audio adapter referenced by hAdapter. All AudioScience adapters have unique serial numbers assigned during manufacturing.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>pdwSerial-Number</i>	The returned serial number.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[adapter/main.c](#).

7.5.2.17 ASX32_API ASX_ERROR ASX_Adapter_ReadNvMem (
ASX_HANDLE hAdapter, const int nAddress, unsigned char *
pcValue)

Read a byte from the non-volatile memory.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>nAddress</i>	The address to read the byte from.
<i>pcValue</i>	A pointer to the returned byte.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.18 ASX32_API ASX_ERROR ASX_Adapter_ReadProperty (
ASX_HANDLE hAdapter, const int nIndex, unsigned short *
pwParm1, unsigned short * pwParm2)

Read an adapter's property value.

Current support property indexes are:

- 1 = ERRATA_1, returns whether errata_1 workaround for 6100 cards is turned on.
- 2 = SSX2_SETTING, returns whether SSX2 is on or off.
- 3 = SYNC_HEADER_CONNECTIONS (read-only), returns the number of headers connected.
- 4 = SUPPORT_SSX2 (read-only), returns true or false.
- 5 = SUPPORTS_FW_UPDATE (read-only), device supports firmware updating
- 6 = FIRMWARE_ID (read-only), firmware ID

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>nIndex</i>	The index of the property to read.
<i>pwParm1</i>	Receives property specific value.
<i>pwParm2</i>	Receives property specific value.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.19 ASX32.API ASX_ERROR ASX_Adapter.SetMode (
ASX_HANDLE hAdapter, const enum asxADAPTERMODE
eMode)

Set the current adapter mode.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>eMode</i>	The adapter mode to set. This must be one of the options returned by calls to ASX_Adapter_EnumerateMode() .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.20 ASX32.API ASX_ERROR ASX_Adapter.WriteNvMem (
ASX_HANDLE hAdapter, const int nAddress, const unsigned
char cValue)

Write a byte to the non-volatile memory.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>nAddress</i>	The address to write the byte to.
<i>cValue</i>	The byte to write.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.5.2.21 ASX32.API ASX_ERROR ASX_Adapter.WriteProperty (
ASX_HANDLE hAdapter, const int nIndex, const unsigned short
wParm1, const unsigned short wParm2)

Write an adapter property value.

Parameters

<i>hAdapter</i>	A handle to an ASX adapter object.
<i>nIndex</i>	The property number.
<i>wParm1</i>	Property specific value.
<i>wParm2</i>	Property specific value.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.6 Mixer functions

The mixer functions are used to access mixer nodes and controls.

Functions

- [ASX_Mixer_ResetControls](#)
This function sets all the controls in the mixer to a known state.
- [ASX_Mixer_GetSourceNodeCount](#)
This function returns the number of source nodes in the mixer.
- [ASX_Mixer_GetSourceNode](#)
This function gets the handle of a particular source node.
- [ASX_Mixer_GetDestinationNodeCount](#)
This function returns the number of destination nodes in the mixer.
- [ASX_Mixer_GetDestinationNode](#)
This function gets the handle of the specified destination node.
- [ASX_Mixer_GetNodeByType](#)
Get a node by type.
- [ASX_Mixer_GetNodeTypeCount](#)
Get the number of nodes of the specified type.
- [ASX_Mixer_GetControlCount](#)
This function returns the total number of controls in the mixer.
- [ASX_Mixer_GetControl](#)
Given a control index, this function returns a handle to the specified control.
- [ASX_Mixer_GetControlByNode](#)
Given source and destination node handles as well as the control type, return the specified control.

- [ASX_Mixer_GetControlByNodeTypeAndIndex](#)

Given source and destination node specifications as well as the control type, return the specified control.

- [ASX_Mixer_GetBlockControlByNodeTypeAndIndex](#)

Given source and destination node specifications as well as the block control name, return the specified control.

7.6.1 Detailed Description

The mixer functions are used to access mixer nodes and controls.

7.6.2 Function Documentation

7.6.2.1 ASX32_API ASX_ERROR ASX_Mixer_GetBlockControlByNodeTypeAndIndex (
ASX_HANDLE hMixer, const enum asxNODE nSourceNodeType,
const int nSourceIndex, const enum asxNODE nDestinationNodeType, const int
nDestinationIndex, const char * pszBlockName, ASX_HANDLE * p_hControlBase)

Given source and destination node specifications as well as the block control name, return the specified control.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>nSourceNodeType</i>	The type of the ASX source node object. Typically this will be set to one of asxNODE . The type may be set to asxNODE_NONE (or 0) if the requested control does not have a source node.
<i>nSourceIndex</i>	The source node index. This may be set to 0 if the requested control does not have a source node.
<i>nDestinationNodeType</i>	The type of the ASX destination node object. Typically this will be set to one of asxNODE . The type may be set to asxNODE_NONE (or 0) if the requested control does not have a destination node.
<i>nDestinationIndex</i>	The destination node index. This may be set to 0 if the requested control does not have a destination node.
<i>pszBlockName</i>	The block control name.
<i>p_hControlBase</i>	The returned ASX control handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.6.2.2 ASX32_API ASX_ERROR ASX_Mixer_GetControl (
ASX_HANDLE *hMixer*, const int *nControl*, ASX_HANDLE *
***p_hControlBase*)**

Given a control index, this function returns a handle to the specified control.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>nControl</i>	The index of the control handle to return.
<i>p_hControlBase</i>	The returned ASX control handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.6.2.3 ASX32_API ASX_ERROR ASX_Mixer_GetControlByNode (
const ASX_HANDLE *hMixer*, const ASX_HANDLE
***hSourceNode*, const ASX_HANDLE *hDestinationNode*, const enum**
***asxCONTROL* *eControlType*, ASX_HANDLE * *p_hControlBase*)**

Given source and destination node handles as well as the control type, return the specified control.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>hSourceNode</i>	A handle to an ASX source node object. This handle may be set to asxNODE_NONE (or 0) if the requested control does not have a source node.
<i>hDestinationNode</i>	A handle to an ASX destination node object. This handle may be set to asxNODE_NONE (or 0) if the requested control does not have a destination node.
<i>eControlType</i>	The control type. Should be set to one of asxCONTROL .
<i>p_hControlBase</i>	The returned ASX control handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[mixer/main.c](#).

7.6.2.4 ASX32_API ASX_ERROR ASX_Mixer.GetControlByNodeTypeAndIndex (
ASX_HANDLE *hMixer*, const enum asxNODE *nSourceNodeType*,
const int *nSourceIndex*, const enum asxNODE *nDestinationNodeType*, const int
***nDestinationIndex*, const enum asxCONTROL *eControlType*, ASX_HANDLE ***
***p_hControlBase*)**

Given source and destination node specifications as well as the control type, return the specified control.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>nSourceNodeType</i>	The type of the ASX source node object. Typically this will be set to one of asxNODE . The type may be set to asxNODE_NONE (or 0) if the requested control does not have a source node.
<i>nSourceIndex</i>	The source node index. This may be set to 0 if the requested control does not have a source node.
<i>nDestinationNodeType</i>	The type of the ASX destination node object. Typically this will be set to one of asxNODE . The type may be set to asxNODE_NONE (or 0) if the requested control does not have a destination node.
<i>nDestinationIndex</i>	The destination node index. This may be set to 0 if the requested control does not have a destination node.
<i>eControlType</i>	The control type. Should be set to one of asxCONTROL .
<i>p_hControlBase</i>	The returned ASX control handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [record/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.6.2.5 ASX32_API ASX_ERROR ASX_Mixer.GetControlCount (
ASX_HANDLE *hMixer*, int * *pnControls*)

This function returns the total number of controls in the mixer.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>pnControls</i>	The returned number of controls.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.6.2.6 ASX32_API ASX_ERROR ASX_Mixer_GetDestinationNode (
ASX_HANDLE hMixer, const int nDestinationNode,
ASX_HANDLE * p_hNode)

This function gets the handle of the specified destination node.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>nDestinationNode</i>	The index of the destination node to get. This should be a number in the range of 0 to the total count returned by ASX_Mixer_GetDestinationNodeCount() .
<i>p_hNode</i>	The returned destination node handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.6.2.7 ASX32_API ASX_ERROR ASX_Mixer_GetDestinationNodeCount (
ASX_HANDLE hMixer, int * pnCount)

This function returns the number of destination nodes in the mixer.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>pnCount</i>	The returned number of destination nodes.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.6.2.8 ASX32_API ASX_ERROR ASX_Mixer_GetNodeByType (
ASX_HANDLE hMixer, const enum asxNODE eType, const int
nIndex, ASX_HANDLE * p_hNode)

Get a node by type.

This function searches all mixer nodes for a node of a particular type and index. This function could be used to find a "LineOut" 1 node for example.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>eType</i>	The node type to get. This should be set to one of asxNODE .
<i>nIndex</i>	The index of the node to get. If the adapter has 4 line outs, for example, and nType is set to asxNODE_LINE_OUT , then the valid range for nIndex would be 0-3.
<i>p_hNode</i>	The returned destination node handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[mixer/main.c](#).

7.6.2.9 ASX32_API ASX_ERROR ASX_Mixer_GetNodeTypeCount (
ASX_HANDLE hMixer, const enum asxNODE eType, int *
pnCount)

Get the number of nodes of the specified type.

This function returns the number of nodes of the specified type in the mixer. For example, an adapter with 4 line outs would return 4 when nType is set to [asxNODE_LINE_OUT](#).

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>eType</i>	The node type to get. This should be set to one of asxNODE .
<i>pnCount</i>	The returned number of nodes of type nType.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[mixer/main.c](#).

7.6.2.10 ASX32_API ASX_ERROR ASX_Mixer_GetSourceNode (
ASX_HANDLE hMixer, const int nSourceNode,
ASX_HANDLE * p_hNode)

This function gets the handle of a particular source node.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>nSourceNode</i>	The index of the source node to get. This should be a number in the range of 0 to the total count returned by ASX_Mixer_GetSourceNodeCount() .
<i>p_hNode</i>	The returned source node handle.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.6.2.11 ASX32_API ASX_ERROR ASX_Mixer_GetSourceNodeCount (ASX_HANDLE hMixer, int * pnCount)

This function returns the number of source nodes in the mixer.

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
<i>pnCount</i>	The returned number of source nodes.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.6.2.12 ASX32_API ASX_ERROR ASX_Mixer_ResetControls (ASX_HANDLE hMixer)

This function sets all the controls in the mixer to a known state.

- SampleClock is set to Local and 44.1kHz
- Volume controls on a single node are set to 0dB
- Channel Modes are set to Normal
- Volume controls between a src and dest node are set to 0dB if the node indexes match, otherwise -100db
- Multiplexers are set to LineIn
- Level controls are set to +14dBu.
- VOX control set to -100dB (off)

Parameters

<i>hMixer</i>	A handle to an ASX mixer object.
---------------	----------------------------------

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.7 Node functions

The node functions are used to access nodes objects.

Functions

- [ASX_Node_GetType](#)

Returns the node type of the given node.

- [ASX_Node_GetIndex](#)

Returns the index of the given node.

- [ASX_Node_GetLocation](#)

Returns the location of the given node in terms of module slots and position on the module that contains the node.

- [ASX_Node_GetSubSystem](#)

Returns the sub system handle of the given node.

- [ASX_Node_GetName](#)

Get the name of the node.

- [ASX_Mixer_GetNodeType](#)

- [ASX_Mixer_GetNodeIndex](#)

7.7.1 Detailed Description

The node functions are used to access nodes objects. The node objects as they are implemented in ASX are really just placeholders. These functions allow an application to query node information, including the node type, index and name. Often these functions will be called after after the source and/or destination nodes of a particular control have been obtained.

7.7.2 Function Documentation

7.7.2.1 ASX32_API ASX_ERROR ASX_Mixer_GetNodeIndex (
ASX_HANDLE hNode, int * pnIndex)

Deprecated

This function has been superseded by [ASX_Node_GetIndex\(\)](#)

7.7.2.2 ASX32_API ASX_ERROR ASX_Mixer_GetNodeType (
ASX_HANDLE hNode, enum asxNODE * peType)

Deprecated

This function has been superseded by [ASX_Node_GetType\(\)](#)

7.7.2.3 ASX32_API ASX_ERROR ASX_Node_GetIndex (
ASX_HANDLE hNode, int * pnIndex)

Returns the index of the given node.

Parameters

<i>hNode</i>	A handle to an ASX node object.
<i>pnIndex</i>	The returned ASX node index. For example, if the hNode object represented the second line out, pnIndex would return 1. Index 0 would be the first line out, so index 1 is the second line out. Also returns 0 if hNode=asxNODE_NONE.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.7.2.4 ASX32_API ASX_ERROR ASX_Node_GetLocation (
ASX_HANDLE hNode, int * pnModuleSlot, int *
pnNodeIndexOnSlot, char * pszModuleName, const int nStringLength)

Returns the location of the given node in terms of module slots and position on the module that contains the node.

Parameters

<i>hNode</i>	A handle to an ASX node object.
--------------	---------------------------------

<i>pnModuleSlot</i>	The returned module's slot number.
<i>pnNodeIndexOnSlot</i>	The returned node's index on the module.
<i>pszModuleName</i>	The returned module name. This will be something like "ASI1441". The string passed in should be of length ASX_SHORT_STRING.
<i>nStringLength</i>	The length of the passed in string.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.7.2.5 ASX32_API ASX_ERROR ASX_Node_GetName (
ASX_HANDLE hNode, char * pszNodeName, const int
nStringLength)

Get the name of the node.

Parameters

<i>hNode</i>	A handle to an ASX node object.
<i>pszNodeName</i>	The string buffer of size nStringLength allocated by the caller.
<i>nStringLength</i>	The length of the pszNodeName buffer. Should be ASX_LONG_STRING.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.7.2.6 ASX32_API ASX_ERROR ASX_Node_GetSubSystem (
ASX_HANDLE hNode, int * p_nSubSystem)

Returns the sub system handle of the given node.

Parameters

<i>hNode</i>	A handle to an ASX node object.
<i>p_nSubSystem</i>	The returned ASX sub system type.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.7.2.7 ASX32 API ASX_ERROR ASX_Node_GetType (ASX_HANDLE hNode, enum asxNODE * peType)

Returns the node type of the given node.

Parameters

<i>hNode</i>	A handle to an ASX node object.
<i>peType</i>	The returned ASX node type. This will below to one of asxNODE . If hNode is null, *peNode=asxNODE_NONE - this is not an error.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.8 Control generic functions

These generic control functions operate on all control objects.

Functions

- [ASX_Control_GetType](#)
Generic control function to get the type of a control.
- [ASX_Control_GetSourceNode](#)
Generic control function to get the source node of a control.
- [ASX_Control_GetDestinationNode](#)
Generic control function to get the destination node of a control.
- [ASX_Control_GetHpiControl](#)
Tunnel through ASX to get HPI control parameters (NOT IMPLEMENTED YET).
- [ASX_Control_GetSubSystem](#)
Returns the sub system handle of the given control.

7.8.1 Detailed Description

These generic control functions operate on all control objects. All ASX control objects have type and source and destination node properties. The generic control functions that follow support querying those properites.

7.8.2 Function Documentation

7.8.2.1 ASX32_API ASX_ERROR ASX_Control.GetDestinationNode (ASX_HANDLE hControl, ASX_HANDLE * p_hNode)

Generic control function to get the destination node of a control.

Parameters

<i>hControl</i>	A handle to an ASX control object.
<i>p_hNode</i>	The destination node ASX object handle. This can be zero if the control does not have a destination node.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.8.2.2 ASX32_API ASX_ERROR ASX_Control.GetHpiControl (ASX_HANDLE hControl, void ** pphHpiSubSys, unsigned int * phHpiControl)

Tunnel through ASX to get HPI control parameters (NOT IMPLEMENTED YET).

Parameters

<i>hControl</i>	A handle to an ASX control object.
<i>pphHpiSubSys</i>	Pointer to an HPI_HSUBSYS object.
<i>phHpiControl</i>	Pointer to an HPI_HCONTROL object.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned. Example

```
HPI_HSUBSYS *phHPISubSys;
HPI_HCONTROL hHPIControl;
ASX_HANDLE hASXControl;    // this is filled in from somewhere

ASX_Control_GetHPIControl( hASXControl, &phHPISubSys, &hHPIControl);
```

7.8.2.3 ASX32_API ASX_ERROR ASX_Control.GetSourceNode (ASX_HANDLE hControl, ASX_HANDLE * p_hNode)

Generic control function to get the source node of a control.

Parameters

<i>hControl</i>	A handle to an ASX control object.
<i>p_hNode</i>	The source node ASX object handle. This can be zero if the control does not have a source node.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), and [mixer/main.c](#).

7.8.2.4 ASX32_API ASX_ERROR ASX_Control_GetSubSystem (ASX_HANDLE hControl, int * p_nSubSystem)

Returns the sub system handle of the given control.

Parameters

<i>hControl</i>	A handle to an ASX control object.
<i>p_nSubSystem</i>	The returned ASX sub system type.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.8.2.5 ASX32_API ASX_ERROR ASX_Control_GetType (ASX_HANDLE hControl, enum asxCONTROL * peControl)

Generic control function to get the type of a control.

Parameters

<i>hControl</i>	A handle to an ASX control object.
<i>peControl</i>	The returned control type will be one of asxCONTROL .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

7.9 Player control functions

These functions support file playback.

Functions

- [ASX_Player_Open](#)
Open a file for playback.
- [ASX_Player_Format_GetString](#)
Get the format of the currently opened file as a string.
- [ASX_Player_Format_GetDetails](#)
Get the format of the currently opened file as individual variables.
- [ASX_Player_Preload](#)
Preloads playback buffers from the given position, ready for playback.
- [ASX_Player_Start](#)
Start playback of a previously opened (and optionally pre-loaded) file.
- [ASX_Player_Pause](#)
Pause playback of the currently playing file.
- [ASX_Player_Stop](#)
Stops playback of the currently playing file.
- [ASX_Player_Wait](#)
Wait for the current file to finish.
- [ASX_Player_Close](#)
Close the current playback file.
- [ASX_Player_GetPosition](#)
Get the current playback position as the offset in bytes, samples or milliseconds from the beginning or end of the file depending on the timescale code used.
- [ASX_Player_SetPosition](#)
Sets the playback to the given position.
- [ASX_Player_GetState](#)
Get the current playback state.
- [ASX_Player_SetTimeScale](#)
Set the playback timescale.

- [ASX_Player_GetTimeScale](#)
Get the playback timescale.
- [ASX_Player_GetFilename](#)
Get the current filename, if any.
- [ASX_Player_SetLoopMode](#)
Set the player to loop or single play mode.
- [ASX_Player_GetLoopMode](#)
Get the current player loop mode.
- [ASX_Player_OpenPlaylist](#)
Open a list of files for playback.
- [ASX_Player_PlaylistStatus](#)
Returns playlist status.
- [ASX_Player_RegisterCallback](#)
Register a callback function that should be called when playback has completed.
- [ASX_Player_PlaylistWait](#)
Wait for the playlist to finish.

7.9.1 Detailed Description

These functions support file playback. The player control transparently supports the playback of several different file formats and compression formats. Formats supported for playback in this version:

asxFILE_FORMAT	_WAV	_RAW
Filename Extension	.WAV	any
_PCM8 or _PCM16	supported	write only
_PCM24 or _PCM32	supported	write only
_PCM32_FLOAT	supported	write only
_MPEG_L2	supported	supported
_MPEG_L3	supported	supported
_MPEG_AACPLUS	not supported yet	not supported yet
_DOLBY_AC2	not supported yet	not supported yet

Note: RAW format playback currently cannot support PCM data formats due to the lack format information in the file. A future version of ASX will include a new playback function to play raw PCM files.

Player State Diagram

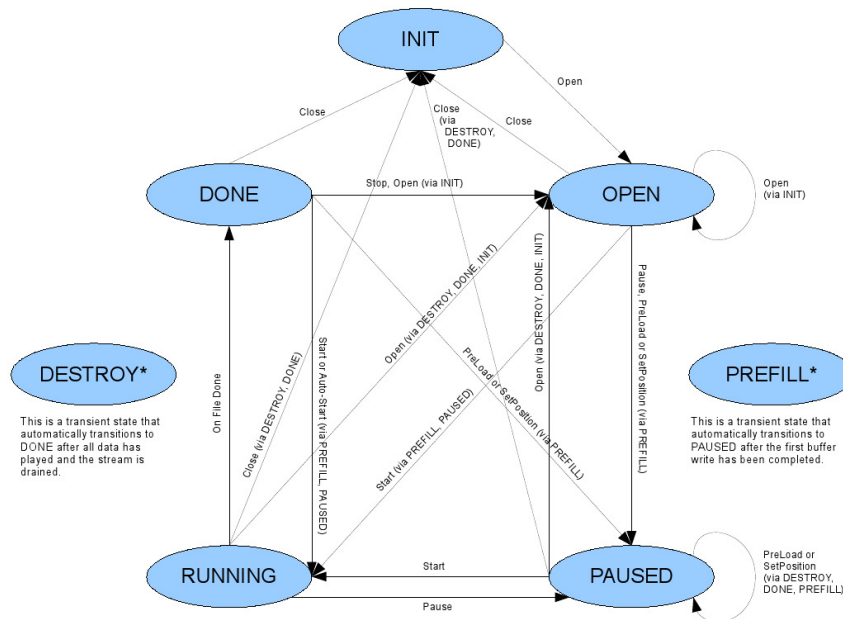


Figure 7.1: ASX Player State Diagram.

7.9.2 Function Documentation

7.9.2.1 ASX32_API ASX_ERROR ASX_Player_Close (ASX_HANDLE hPlayer)

Close the current playback file.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
----------------	---

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#), and [playlist/main.c](#).

7.9.2.2 ASX32_API ASX_ERROR ASX_Player.Format_GetDetails (
**ASX_HANDLE *hPlayer*, enum asxAUDIO_FORMAT *
peFormat, int * *pnChannels*, int * *pnSampleRate*, int * *pnBitRate*)**

Get the format of the currently opened file as individual variables.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>peFormat</i>	The returned format code - see asxAUDIO_FORMAT .
<i>pnChannels</i>	The returned number of channels.
<i>pnSampleRate</i>	The returned sample rate.
<i>pnBitRate</i>	The returned bitrate.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.9.2.3 ASX32_API ASX_ERROR ASX_Player.Format_GetString (
ASX_HANDLE *hPlayer*, char ** *pszFormat*)

Get the format of the currently opened file as a string.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>pszFormat</i>	The returned pointer to a format string.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#), and [playlist/main.c](#).

7.9.2.4 ASX32_API ASX_ERROR ASX_Player.GetFilename (
**ASX_HANDLE *hPlayer*, char * *pszFilename*, const int
nStringLength, int * *pnRequiredLength*)**

Get the current filename, if any.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>pszFilename</i>	The returned pointer to the filename.
<i>nStringLength</i>	The length in bytes of <i>pszFilename</i> .

<i>pnRequiredLength</i>	The required length in bytes of pszRevision.
-------------------------	--

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned. Returns [asxERROR_INDEX_OUT_OF_RANGE](#) if the buffer is too small.

7.9.2.5 ASX32_API ASX_ERROR ASX_Player_GetLoopMode (
ASX_HANDLE hPlayer, int * pnLooping)

Get the current player loop mode.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>pnLooping</i>	Pointer to return value: 1 for looping, 0 for single play.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned. Returns [asxERROR_INDEX_OUT_OF_RANGE](#) if the buffer is too small.

7.9.2.6 ASX32_API ASX_ERROR ASX_Player_GetPosition (
ASX_HANDLE hPlayer, const enum asxTIMESCALE nType,
unsigned long * plPosition)

Get the current playback position as the offset in bytes, samples or milliseconds from the beginning or end of the file depending on the timescale code used.

NOTE: For compressed file it is assumed that the same bitrate is used throughout the file. If the file uses different bitrates (sometimes called "bitrate switching") then the position may not be accurate.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>nType</i>	The units to return the position in (see asxTIMESCALE).
<i>plPosition</i>	The current relative playback position.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.9.2.7 ASX32_API ASX_ERROR ASX_Player.GetState (
ASX_HANDLE *hPlayer*, enum asxPLAYER_STATE * *pnState*
)

Get the current playback state.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>pnState</i>	The current player state. See asxPLAYER_STATE .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#).

7.9.2.8 ASX32_API ASX_ERROR ASX_Player.GetTimeScale (
ASX_HANDLE *hPlayer*, float * *pfTimeScale*)

Get the playback timescale.

This function allows the user read the time ratio. Time scale range is 0.8 - 1.2 (80% to 120%) of original file time.

Note

This function is only supported on the ASI6xxx series adapters.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>pfTimeScale</i>	Pointer to the returned time scale. Range is $0.8 < fTimeScale < 1.2$.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.9.2.9 ASX32_API ASX_ERROR ASX_Player.Open (
ASX_HANDLE *hPlayer*, const char * *pszFile*)

Open a file for playback.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
----------------	---

<i>pszFile</i>	The name of the file to play. This can be the name of a file in the current directory or a fully qualified path name. The file can be a .wav or .mp3 file. Note that MP3 playback is currently only supported on those adapters that support MP3 decoding. This would be the ASI43xx and ASI6xxx series.
----------------	--

You can also generate a sine wave. The format of the string is: "~w,c,f,a,m,s,t"

w = waveform = SINE (default=SINE)

c = channels = 1..8 (default = 2)

f = frequency = 1000 for 1kHz (default=1000)

a = amplitude = -1 for -1dBFS (default=0dBFS, ie full scale)

m = channel_mask = 10 for left only, 01 for right only, 11 for stereo etc (default=1 for all channels)

t = samplotype = (PCM8,PCM16,PCM24,PCM32,FLOAT32), (default=FLOAT32)

s = samplerate = positive integer (default=48000) [validity depends on adapter]

Defaults can be used if the complete string is not specified, ie

"~" -> "~wSINE,c2,f1000,a0,m11,s48000,tFLOAT32"

Any subset of the options may be specified, the remaining options will be set to the defaults. eg "~f500" -> 500Hz stereo sine wave at 0dBFS, 48kHz samplerate

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#), and [playlist/main.c](#).

7.9.2.10 ASX32 API ASX_ERROR ASX_Player.OpenPlaylist (
ASX_HANDLE hPlayer, const char ** pszFileList, const
unsigned int nFiles)

Open a list of files for playback.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>pszFileList</i>	The names of the files to play. This can be the name of a file in the current directory or a fully qualified path name. See ASX_Player_Open() . Note that the player makes a copy of the filelist, so there is no need for the calling application to keep the list of strings in the calling context.
<i>nFiles</i>	The number of files in the playlist.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[playlist/main.c](#).

7.9.2.11 ASX32_API ASX_ERROR ASX_Player_Pause (
ASX_HANDLE hPlayer)

Pause playback of the currently playing file.

Use [ASX_Player_Start](#) to continue playing. To end a paused recording call [ASX_Player_Stop](#).

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
----------------	---

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.9.2.12 ASX32_API ASX_ERROR ASX_Player_PlaylistStatus (
ASX_HANDLE hPlayer, unsigned int * nTotalFileCount, int *
nCurrentFile, char ** szCurrentFilename, unsigned int * nTotalTime_ms, unsigned int
*** nCurrentTime_ms)**

Returns playlist status.

Use this function to monitor playlist progress.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>nTotalFile-Count</i>	Returns the total number of files remaining in the current playlist. This will equal the parameter "nFiles" used in the call to ASX_Player_OpenPlaylist()
<i>nCurrent-File</i>	Returns the index of the currently playing file. A value of -1 indicates that the first file has not yet started. The range of nCurrentFile is 0 to (nTotalFileCount-1). A pause operation does not affect the value returned by nCurrentFile.
<i>szCurrent-Filename</i>	The name of the current file being played. This returns "undefined" before the first Start() command is issued.
<i>nTotalTime_-ms</i>	The total time in milliseconds of all the files that are in the playback list. Note that this variable will be affected if timescaling is enabled.
<i>nCurrentTime_ms</i>	The current accumulated time of the file list playback in milliseconds. This will range from 0 to nTotalTime_ms. Note that this variable will be affected if timescaling is enabled.

Examples:

[playlist/main.c](#).

7.9.2.13 ASX32 API ASX_ERROR ASX_Player_PlaylistWait (
ASX_HANDLE *hPlayer*)

Wait for the playlist to finish.

This function does not return until the current playlist has finished playing.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
----------------	---

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[playlist/main.c](#).

7.9.2.14 ASX32 API ASX_ERROR ASX_Player_PreLoad (
ASX_HANDLE *hPlayer*, const enum asxTIMESCALE *nType*,
const unsigned long *lPosition*)

Preloads playback buffers from the given position, ready for playback.

This function will seek to the specified file position and then load audio buffers from the file. This shortens the time between the time a [ASX_Player_Start\(\)](#) is issued and the time for audio to be output.

Note

This function does not have to be used. It is optional. If [ASX_Player_Start\(\)](#) is called without calling [ASX_Player_PreLoad\(\)](#), the preload operation will happen internal to the [ASX_Player_Start\(\)](#) call. The assumption will also be that playback begins at the start of the audio file.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>nType</i>	The units of <i>lPosition</i> (see asxTIMESCALE).
<i>lPosition</i>	The position to start playback from. A value of zero start from the beginning of the file.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#), and [playlist/main.c](#).

7.9.2.15 ASX32_API ASX_ERROR ASX_Player_RegisterCallback (
ASX_HANDLE *hPlayer*, ASX_PLAYER_CALLBACK *
***pCallback*, const enum asxPLAYER_FLAGS *flags*, void * *pUser1*)**

Register a callback function that should be called when playback has completed.

Note that using this function is optional.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>pCallback</i>	A pointer to a callback of type ASX_PLAYER_CALLBACK.
<i>flags</i>	Defines whether the callback should be called on file completion and/or filelist completion. See asxPLAYER_FLAGS .
<i>pUser1</i>	A user defined pointer that is passed back when a callback is made.

Examples:

[playlist/main.c](#).

7.9.2.16 ASX32_API ASX_ERROR ASX_Player_SetLoopMode (
ASX_HANDLE *hPlayer*, const int *nLooping*)

Set the player to loop or single play mode.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>nLooping</i>	1 for looping, 0 for single play.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned. Returns [asxERROR_INDEX_OUT_OF_RANGE](#) if the buffer is too small.

7.9.2.17 ASX32_API ASX_ERROR ASX_Player_SetPosition (
ASX_HANDLE *hPlayer*, const enum asxTIMESCALE *nType*,
const unsigned long *lPosition*)

Sets the playback to the given position.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>nType</i>	The units of lPosition (see asxTIMESCALE).
<i>lPosition</i>	The position playback from.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.9.2.18 ASX32_API ASX_ERROR ASX_Player.SetTimeScale (
ASX_HANDLE hPlayer, const float fTimeScale)

Set the playback timescale.

This function allows the user to adjust the time a file takes to playback without affecting the pitch. Time scale range is 0.8 - 1.2 (80% to 120%) of original file time.

Note

This function is only supported on the ASI6xxx series adapters.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
<i>fTimeScale</i>	The time scale to use on the playing file. Range is 0.8 < fTimeScale < 1.2.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.9.2.19 ASX32_API ASX_ERROR ASX_Player.Start (
ASX_HANDLE hPlayer)

Start playback of a previously opened (and optionally pre-loaded) file.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
----------------	---

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#), and [playlist/main.c](#).

7.9.2.20 ASX32_API ASX_ERROR ASX_Player.Stop (
ASX_HANDLE hPlayer)

Stops playback of the currently playing file.

This call resets the play position as well as stopping playback. Use [ASX_Player_Pause\(\)](#) to retain the current position.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
----------------	---

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#).

7.9.2.21 ASX32_API ASX_ERROR ASX_Player_Wait (ASX_HANDLE hPlayer)

Wait for the current file to finish.

This function does not return until the current file has finished playing.

Parameters

<i>hPlayer</i>	A handle to an ASX player control object.
----------------	---

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[play/main.c](#).

7.10 Recorder control functions

These functions support file recording.

Functions

- [ASX_Recorder_Open](#)
Opens the recorder using the specified format.
- [ASX_Recorder_Start](#)
Starts the recording.
- [ASX_Recorder_Stop](#)
Stops the recording.
- [ASX_Recorder_Pause](#)
Pauses the recording.

- [ASX_Recorder_Close](#)

Closes the recording file.

- [ASX_Recorder_GetPosition](#)

Gets the current record position.

- [ASX_Recorder_GetState](#)

Get the current record state.

- [ASX_Recorder_GetFilename](#)

Get the current filename, if any.

- [ASX_Recorder_EnumerateFormat](#)

Enumerates supported recorder formats.

7.10.1 Detailed Description

These functions support file recording. Formats supported for recording in this version:

asxFILE_FORMAT	_WAV	_RAW
Filename Extension	.WAV	any
_PCM8 or _PCM16	supported	write only
_PCM24 or _PCM32	supported	write only
_PCM32_FLOAT	supported	write only
_MPEG_L2	supported	supported
_MPEG_L3	supported	supported
_MPEG_AACPLUS	not supported yet	not supported yet
_DOLBY_AC2	not supported yet	not supported yet

Note: A file recorded using _MPEG_L2 or _MPEG_L3 data format and the _RAW file format will comply with the standard for .MP3 files since the additional header information (i.e. ID3 header) is optional.

Recorder State Diagram

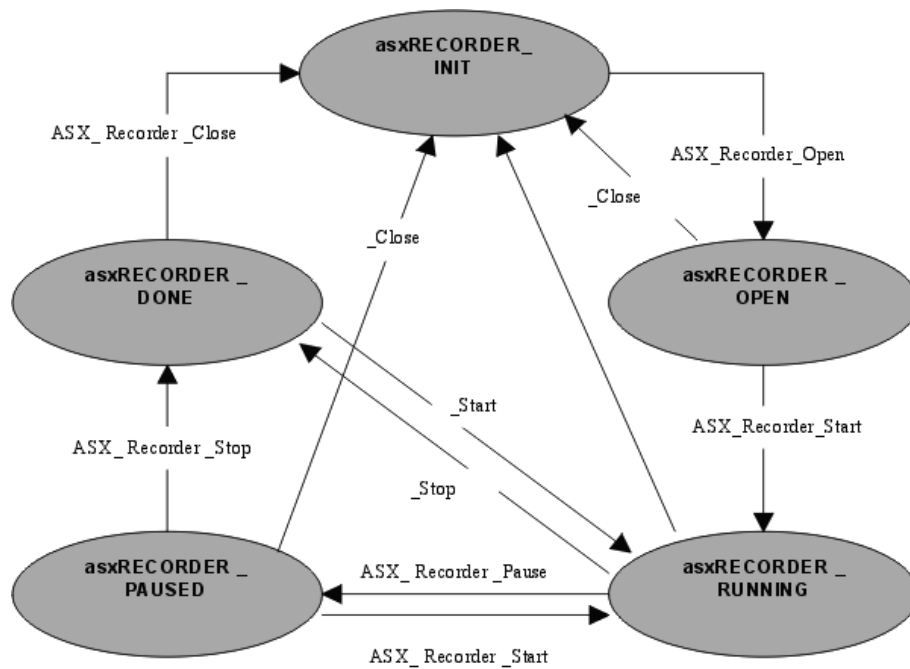


Figure 7.2: ASX Player State Diagram.

7.10.2 Function Documentation

7.10.2.1 ASX32_API ASX_ERROR ASX_Recorder_Close (ASX_HANDLE hRecorder)

Closes the recording file.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
------------------	-------------------------------------

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[dual_mono_record/main.c](#), and [record/main.c](#).

7.10.2.2 ASX32 API ASX_ERROR ASX_Recorder_EnumerateFormat (
ASX_HANDLE *hRecorder*, const int *nIndex*, enum
asxAUDIO_FORMAT * *peFormat*, int * *pnCount*)

Enumerates supported recorder formats.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
<i>nIndex</i>	The format number.
<i>peFormat</i>	Returned enumerated format (See asxFORMAT).
<i>pnCount</i>	Returned total numebr of formats.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.10.2.3 ASX32 API ASX_ERROR ASX_Recorder_GetFilename (
ASX_HANDLE *hRecorder*, char * *pszFilename*, const int
***nStringLength*, int * *pnRequiredLength*)**

Get the current filename, if any.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
<i>pszFilename</i>	The returned pointer to the filename.
<i>nStringLength</i>	The length in bytes of pszFilename.
<i>pnRequiredLength</i>	The required length in bytes of pszRevision.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned. Returns asxERROR_INDEX_OUT_OF_RANGE if the buffer is too small.

7.10.2.4 ASX32 API ASX_ERROR ASX_Recorder_GetPosition (
ASX_HANDLE *hRecorder*, const enum asxTIMESCALE
***nType*, unsigned long * *plPosition*)**

Gets the current record position.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
<i>nType</i>	The units to return the position in (see asxTIMESCALE).
<i>plPosition</i>	The current record position.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.10.2.5 ASX32_API ASX_ERROR ASX_Recorder_GetState (
ASX_HANDLE *hRecorder*, enum asxRECORDER_STATE *
***peState*)**

Get the current record state.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
<i>peState</i>	The current recorder state. See asxRECORDER_STATE .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.10.2.6 ASX32_API ASX_ERROR ASX_Recorder_Open (
ASX_HANDLE *hRecorder*, const char * *pszFile*, const enum
asxFILE_FORMAT *nFileType*, const enum asxFILE_MODE *nFileMode*,
const int *nChannels*, const enum asxAUDIO_FORMAT *nFormat*, const long
***lSampleRate*, const long *lBitrate*, const enum asxRECORD_MODE *nMode*)**

Opens the recorder using the specified format.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
<i>pszFile</i>	The name of the file to be opened.
<i>nFileType</i>	File format. This specifies the header information (if any) to be placed on the audio file. See asxFILE_FORMAT for the complete range of formats.
<i>nFileMode</i>	Sets the mode for opening an existing file for recording. See asxFILE_MODE .
<i>nChannels</i>	The number of channels. Currently either 1 or 2 channels are supported.
<i>nFormat</i>	Audio format is used to specified the format of the recorded samples. See asxAUDIO_FORMAT for a complete list of formats. Note that not all formats are supported on all adapters, so it is important to check for errors after making this call.
<i>lSampleRate</i>	The sample rate should be set to 8000-192000 Hz. Note that some adapters do not support all sample rates, so it is important to check for errors after making this call.
<i>lBitrate</i>	Bitrate = 8000 to 384000 bps (MPEG only)
<i>nMode</i>	Recording mode applies to MPEG only. See asxRECORD_MODE .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[dual_mono_record/main.c](#), and [record/main.c](#).

7.10.2.7 ASX32_API ASX_ERROR ASX_Recorder_Pause (
ASX_HANDLE *hRecorder*)

Pauses the recording.

Use [ASX_Recorder_Start](#) to continue recording.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
------------------	-------------------------------------

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[record/main.c](#).

7.10.2.8 ASX32_API ASX_ERROR ASX_Recorder_Start (
ASX_HANDLE *hRecorder*)

Starts the recording.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
------------------	-------------------------------------

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[dual_mono_record/main.c](#), and [record/main.c](#).

7.10.2.9 ASX32_API ASX_ERROR ASX_Recorder_Stop (
ASX_HANDLE *hRecorder*)

Stops the recording.

Parameters

<i>hRecorder</i>	A handle to an ASX recorder object.
------------------	-------------------------------------

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[dual_mono_record/main.c](#), and [record/main.c](#).

7.11 Meter control functions

These functions support reading peak meter information.

Functions

- [ASX_Meter_GetChannels](#)
Returns the number of channels this peak meter has.
- [ASX_Meter_GetPeak](#)
Returns the peak meter reading for the given meter control.
- [ASX_Meter_GetRMS](#)
Returns the RMS meter reading for the given meter control.
- [ASX_Meter_SetBallistics](#)
Set the meter ballistics.
- [ASX_Meter_GetBallistics](#)
Get meter ballistics.

7.11.1 Detailed Description

These functions support reading peak meter information.

7.11.2 Function Documentation

7.11.2.1 ASX32_API ASX_ERROR ASX_Meter_GetBallistics (
ASX_HANDLE *hMeter*, const enum asxMETER_TYPE
***nMeterType*, float * *fAttackTimeMs*, float * *fDecayTimeMs*)**

Get meter ballistics.

Parameters

<i>hMeter</i>	A handle to an ASX meter object.
<i>nMeterType</i>	Which meter part to get ballistics settings from. See asxMETER_TYPE .

<i>fAttack-TimeMs</i>	The attack time in milliseconds.
<i>fDecay-TimeMs</i>	The decay time in milliseconds.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.11.2.2 ASX32 API ASX_ERROR ASX_Meter_GetChannels (
ASX_HANDLE hMeter, int * pnChannels)

Returns the number of channels this peak meter has.

Parameters

<i>hMeter</i>	A handle to an ASX meter object.
<i>pnChannels</i>	The returned number of channels.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.11.2.3 ASX32 API ASX_ERROR ASX_Meter_GetPeak (
ASX_HANDLE hMeter, float * fdB, const int nChannels)

Returns the peak meter reading for the given meter control.

Some adapters (ASI6000,ASI5000) implement meter controls with ballistics. This means that instead of following the signal instantaneously, meters values have finite attack and decay time constants Ta and Td. For instance when the input is removed, the meter value will decay towards zero (whether or not the meter control is read). It will decay to 37% of its original value in Td seconds. (14% @ 2xTd, 5% @ 3xTd etc)

If meter ballistics are not implemented, when the ASX_Meter_GetPeak function is called the meter statistic for that control will be reset to zero. This means that the period over which the statistic is computed depends on the time between calls to [ASX_Meter_GetPeak\(\)](#). If the call is made 100 times a second, then the value returned would represent the peak during a 10ms interval of audio.

The lower limit of meter return value, depends on the adapter series. For all ASI4000 adapters, the minimum value returned is -100dB. For ASI6000,ASI5000 adapters, the minimum value returned is -192dB.

Parameters

<i>hMeter</i>	A handle to an ASX meter object.
<i>fdB</i>	A pointer to receive the peak meter reading in dB. The range of this reading is 0.0 to -120.0 dB.
<i>nChannels</i>	The number of channels. This should match the number of elements in the fGetDb array.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.11.2.4 ASX32_API ASX_ERROR ASX_Meter_GetRMS (ASX_HANDLE *hMeter*, float * *fdB*, const int *nChannels*)

Returns the RMS meter reading for the given meter control.

The peak is stereo and the units are in decibels relative to full-scale digital (dbFs). The RMS measurement depends on the waveform shape. For example, playing a fullscale sine wave (with an amplitude of +/-32767 for a 16bit PCM format) will return a RMS reading of -3dB (compared with a Peak meter reading of 0dbFs), while playing a square wave will return a RMS reading of 0dB and playing an impulsive signal like solo drums will return an RMS value much lower than the peak value.

If this meter has ballistics, then Ta=Td=150ms, which simulates the ballistics of a VU meter.

Parameters

<i>hMeter</i>	A handle to an ASX meter object.
<i>fdB</i>	A pointer to receive the RMS meter reading in dB. The range of this reading is 0.0 to -120.0 dB.
<i>nChannels</i>	The number of channels. This should match the number of elements in the fGetDb array.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.11.2.5 ASX32_API ASX_ERROR ASX_Meter_SetBallistics (ASX_HANDLE *hMeter*, const enum [asxMETER_TYPE](#) *nMeterType*, const float *fAttackTimeMs*, const float *fDecayTimeMs*)

Set the meter ballistics.

The attack and decay values represent the time constants of the equivalent single pole low pass filter used to create the ballistics. With a time constant of T, if the meter is stable at full scale and the input is suddenly removed, the meter will decay. Similarly, if the meter is at zero and a full scale input is applied will move to the new reading at a rate specified by the attack time constant.

Driver versions up to and including version 4.04.xx implement a single global ballistics setting for all meters, i.e. if you change the ballistics on one meter, the ballistics on all meters are updated. Driver versions in the 4.05.xx series and later implement independent ballistics for each meter.

The following table shows the percentage of the final meter value over time, when a constant input is suddenly removed (decay) or applied (attack):

<i>Time</i>	<i>Meter decay</i>	<i>Meter attack</i>
0	100%	0%
T	37%	63%
2T	14%	86%
3T	5%	95%
4T	2%	98%
5T	0.7%	99%

Table 7.97: Attack and decay compared to time.

The table was calculated using the following formulas:

$$decay = initial * e^{-\frac{t}{T}}$$

and

$$attack = finale * (1 - e^{-\frac{t}{T}})$$

Parameters

<i>hMeter</i>	A handle to an ASX meter object.
<i>nMeterType</i>	The meter type to set the ballistics of. See asxMETER_TYPE .
<i>fAttack-TimeMs</i>	The attack time in milliseconds.
<i>fDecay-TimeMs</i>	The decay time in milliseconds.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.12 Volume control functions

These functions support volume control manipulation.

Functions

- [ASX_Volume_GetChannels](#)
Returns the number of channels this volume control has.
- [ASX_Volume_SetMute](#)
Sets mute for this volume control.
- [ASX_Volume_GetMute](#)
Returns the mute setting for this volume control.
- [ASX_Volume_SetGain](#)
Set volume.
- [ASX_Volume_GetGain](#)
Get volume.
- [ASX_Volume_GetRange](#)
Get that range of volume settings available.
- [ASX_Volume_SetAutofade](#)
Set an autofade operation.

7.12.1 Detailed Description

These functions support volume control manipulation.

7.12.2 Function Documentation

7.12.2.1 ASX32_API ASX_ERROR ASX_Volume_GetChannels (ASX_HANDLE hVolume, int * pnChannels)

Returns the number of channels this volume control has.

Parameters

<i>hVolume</i>	A handle to an ASX volume object.
<i>pnChannels</i>	The returned number of channels.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[volume/main.c](#).

7.12.2.2 ASX32 API ASX_ERROR ASX_Volume_GetGain (ASX_HANDLE *hVolume*, float * *fdB*, const int *nChannels*)

Get volume.

Parameters

<i>hVolume</i>	A handle to an ASX volume control.
<i>fdB</i>	The returned gain in dBFS, i.e. 0dB is fullscale.
<i>nChannels</i>	The number of channels. This should match the number of elements in the fGetGain array.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[dual_mono_play/main.c](#).

7.12.2.3 ASX32 API ASX_ERROR ASX_Volume_GetMute (ASX_HANDLE *hVolume*, int * *mute*, const int *nChannels*)

Returns the mute setting for this volume control.

All returned settings will contain the same value.

Parameters

<i>hVolume</i>	A handle to an ASX volume object.
<i>mute</i>	The returned mute setting. 1 is mute, 0 is unmuted.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.12.2.4 ASX32 API ASX_ERROR ASX_Volume_GetRange (ASX_HANDLE *hVolume*, float * *fMinGain*, float * *fMaxGain*, float * *fGainStep*)

Get that range of volume settings available.

Parameters

<i>hVolume</i>	A handle to an ASX volume control.
<i>fMinGain</i>	The returned minimum gain in dBFS, i.e. 0dB is fullscale.
<i>fMaxGain</i>	The returned maximum gain in dBFS, i.e. 0dB is fullscale.
<i>fGainStep</i>	The returned gain stepsize in dB.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[volume/main.c](#).

7.12.2.5 ASX32_API ASX_ERROR ASX_Volume_SetAutofade (
ASX_HANDLE hVolume, const float * fSetdB,
const int nChannels, const ASX_TIME nDuration, const enum
asxVOLUME_AUTOFADE eProfile)

Set an autofade operation.

This function commands the adapter to automatically fade the volume to the specified volume setting. The autofade operation begins when the command is issued.

Parameters

<i>hVolume</i>	A handle to an ASX volume control.
<i>fSetdB</i>	The target gain in dBFS, i.e. 0dB is fullscale.
<i>nChannels</i>	The number of channels. This should match the number of elements in the fGetGain array.
<i>nDuration</i>	The duration in milliseconds.
<i>eProfile</i>	The fading profile. See asxVOLUME_AUTOFADE . All ASI adapters support asxVOLUME_AUTOFADE_LINEAR .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.12.2.6 ASX32_API ASX_ERROR ASX_Volume_SetGain (
ASX_HANDLE hVolume, float * fSetdB, const int nChannels)

Set volume.

Parameters

<i>hVolume</i>	A handle to an ASX volume control.
<i>fSetdB</i>	The gain to set in dBFS, i.e. 0dB is fullscale.
<i>nChannels</i>	The number of channels. This should match the number of elements in the fSetGain array.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[dual_mono_play/main.c](#), and [volume/main.c](#).

7.12.2.7 ASX32 API ASX_ERROR ASX_Volume_SetMute (ASX_HANDLE hVolume, int * mute, const int nChannels)

Sets mute for this volume control.

All channels must contain the same setting.

Parameters

<i>hVolume</i>	A handle to an ASX volume object.
<i>mute</i>	The mute setting. 1 is mute, 0 is unmuted.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.13 Level control functions

These functions support level/trim control manipulation in line ins and line outs.

Functions

- [ASX_Level_Set](#)
Set the analog input or output level (sometimes called trim).
- [ASX_Level_Get](#)
Get the analog input or output level (sometimes called trim).
- [ASX_Level_GetRange](#)
Get that range of level settings available.

7.13.1 Detailed Description

These functions support level/trim control manipulation in line ins and line outs.

7.13.2 Function Documentation

7.13.2.1 ASX32 API ASX_ERROR ASX_Level_Get (ASX_HANDLE hLevel, float * fGain)

Get the analog input or output level (sometimes called trim).

Parameters

<i>hLevel</i>	A handle to an ASX level control.
<i>fGain</i>	The returned level reading. The level has the units of dBu.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.13.2.2 ASX32_API ASX_ERROR ASX_Level.GetRange (
ASX_HANDLE *hLevel*, float * *fMinGain*, float * *fMaxGain*, float *
***fGainStep*)**

Get that range of level settings available.

Parameters

<i>hLevel</i>	A handle to an ASX level control.
<i>fMinGain</i>	The returned minimum gain in dBFS, i.e. 0dB is fullscale.
<i>fMaxGain</i>	The returned maximum gain in dBFS, i.e. 0dB is fullscale.
<i>fGainStep</i>	The returned gain stepsize in dB.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.13.2.3 ASX32_API ASX_ERROR ASX_Level.Set (
ASX_HANDLE *hLevel*, const float *fGain*)

Set the analog input or output level (sometimes called trim).

Parameters

<i>hLevel</i>	A handle to an ASX level control.
<i>fGain</i>	The level to set. The level has the units of dBu. The typical range of settings for most AudioScience adapters is -10 to +26 dBu.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.14 Multiplexer control functions

These functions support multiplexer type operations.

Functions

- [ASX_Multiplexer_Enumerate](#)
Enumerate each multiplexer option.
- [ASX_Multiplexer_Get](#)

Get the current multiplexer setting.

- [ASX_Multiplexer_Set](#)

Set the multiplexer.

7.14.1 Detailed Description

These functions support multiplexer type operations.

7.14.2 Function Documentation

7.14.2.1 ASX32_API ASX_ERROR ASX_Multiplexer_Enumerate (
ASX_HANDLE hMux, const int nIndex, enum asxNODE *
peNode, int * pnNodeIndex, int * pnCount)

Enumerate each multiplexer option.

Returns each multiplexer option in terms of a node type and an index. For example, if value selections are Line Ins 1-4, this function will return [asxNODE_LINE_IN](#) with the nIndex value set to 0-3. To find all available settings, this function should be called repeatedly with incrementing values on nIndex until an error is returned.

Parameters

<i>hMux</i>	A handle to an ASX multiplexer control.
<i>nIndex</i>	The index of the multiplexer option to fetch.
<i>peNode</i>	The returned multiplexer node. See asxNODE for available nodes.
<i>pnNodeIndex</i>	The returned node index.
<i>pnCount</i>	The total number of available multiplexer settings.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[mux/main.c](#).

7.14.2.2 ASX32_API ASX_ERROR ASX_Multiplexer_Get (
ASX_HANDLE hMux, enum asxNODE * peNode, int *
pnNodeIndex)

Get the current multiplexer setting.

Parameters

<i>hMux</i>	A handle to an ASX multiplexer control.
-------------	---

<i>peNode</i>	The returned node. See asxNODE for available nodes.
<i>pnNodeIndex</i>	The returned node index.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[mux/main.c](#).

7.14.2.3 ASX32_API ASX_ERROR ASX_Multiplexer_Set (
ASX_HANDLE hMux, const enum asxNODE eNode, const int
nNodeIndex)

Set the multiplexer.

Parameters

<i>hMux</i>	A handle to an ASX multiplexer control.
<i>eNode</i>	The node to set the multiplexer to. See asxNODE for available nodes.
<i>nNodeIndex</i>	The node index to set.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[dual_mono_record/main.c](#).

7.15 Channel Mode control functions

These functions support channel mode operations that can be used to swap left and right audio channels, or convert stereo sources to mono outputs etc.

Functions

- [ASX_ChannelMode_Enumerate](#)
Enumerate each channel mode option.
- [ASX_ChannelMode_Get](#)
Get the current channel mode.
- [ASX_ChannelMode_Set](#)

7.15.2.3 ASX32_API ASX_ERROR ASX_ChannelMode_Set (
ASX_HANDLE *hMode*, const enum asxCHANNELMODE
***eMode*)**

Set the current channel mode.

Parameters

<i>hMode</i>	A handle to an ASX channel mode control.
<i>eMode</i>	The channel mode to set. This must be one of the options returned by calls to ASX_ChannelMode_Enumerate() .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[dual_mono_play/main.c](#), and [dual_mono_record/main.c](#).

7.16 Tuner control functions

These functions support tuner operations.

Functions

- [ASX_Tuner_EnumerateBand](#)
Enumerate each tuner band option.
- [ASX_Tuner_GetBand](#)
Get the tuner band.
- [ASX_Tuner_SetBand](#)
Set the tuner band.
- [ASX_Tuner_SetFrequency](#)
Set the tuner frequency.
- [ASX_Tuner_GetFrequency](#)
Get the tuner frequency.
- [ASX_Tuner_GetFrequencyRange](#)
Get the tuner frequency range in Hz.
- [ASX_Tuner_GetGainRange](#)
Get the tuner gain range (in dB).

- [ASX_Tuner_SetGain](#)
Set the tuner gain.
- [ASX_Tuner_GetGain](#)
Get the tuner gain.
- [ASX_Tuner_GetRFLevel](#)
Get the tuner RF level.
- [ASX_Tuner_GetRawRFLevel](#)
Get the Raw tuner RF level.
- [ASX_Tuner_GetStatus](#)
Get the tuner status.
- [ASX_Tuner_GetMode](#)
Gets the tuner mode.
- [ASX_Tuner_SetMode](#)
Sets the tuner mode.
- [ASX_Tuner_EnumerateDeemphasis](#)
Enumerates tuner de-emphasis options.
- [ASX_Tuner_SetDeemphasis](#)
Set tuner de-emphasis.
- [ASX_Tuner_GetDeemphasis](#)
Get tuner de-emphasis.
- [ASX_Tuner_EnumerateProgram](#)
Enumerates tuner program options.
- [ASX_Tuner_SetProgram](#)
Set tuner program.
- [ASX_Tuner_GetProgram](#)
Get tuner program.
- [ASX_Tuner_GetHdRadioSignalQuality](#)
- [ASX_Tuner_GetDigitalSignalQuality](#)
Get digital signal quality.
- [ASX_Tuner_GetHdRadioSdkVersion](#)
- [ASX_Tuner_GetHdRadioDspVersion](#)
- [ASX_Tuner_GetFirmwareVersion](#)

- ASX_Tuner_EnumerateHdBlend

- ASX_Tuner_SetHdBlend

- ASX_Tuner_GetHdBlend

- ASX_Tuner_GetDabMultiplexName

- ASX_Tuner_GetDabMultiplexId

- ASX_Tuner_GetDabAudioServiceCount

- ASX_Tuner_GetDabAudioServiceName

- ASX_Tuner_SetDabAudioService

- ASX_Tuner_GetDabServiceId

- ASX_Tuner_GetDabAudioInfo

Generated on Fri Sep 2 2016 10:32:43 for ASX by Doxygen

<i>hTuner</i>	A handle to an ASX tuner control.
<i>nIndex</i>	The index of the tuner band option to fetch.
<i>peBand</i>	The returned tuner band option.
<i>pnCount</i>	The total number of available tuner bands.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.16.2.2 ASX32_API ASX_ERROR ASX_Tuner_EnumerateDeemphasis (
ASX_HANDLE hTuner, const int nIndex, enum
asxTUNERDEEMPHASIS * peDeemphasis, int * pnCount)

Enumerates tuner de-emphasis options.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>nIndex</i>	The number of the de-emphasis setting to retrieve.
<i>peDeemphasis</i>	The de-emphasis option.
<i>pnCount</i>	The total number of de-emphasis options.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.3 ASX32_API ASX_ERROR ASX_Tuner_EnumerateHdBlend (
ASX_HANDLE hTuner, const int nIndex, enum
asxTUNERHDBLEND * peBlend, int * pnCount)

Enumerates tuner blend options.

The API only supports HDRadio in the USA.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>nIndex</i>	The number of the blend settings to retrieve.
<i>peBlend</i>	The program option.
<i>pnCount</i>	The total number of program options.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.4 ASX32_API ASX_ERROR ASX_Tuner_EnumerateProgram (

**ASX_HANDLE *hTuner*, const int *nIndex*, enum
asxTUNERPROGRAM * *peProgram*, int * *pnCount*)**

Enumerates tuner program options.

The API only supports HDRadio in the USA.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>nIndex</i>	The number of the program setting to retrieve.
<i>peProgram</i>	The program option.
<i>pnCount</i>	The total number of program options.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.5 ASX32_API ASX_ERROR ASX_Tuner_GetBand (

ASX_HANDLE *hTuner*, enum asxTUNERBAND * *peBand*)

Get the tuner band.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>peBand</i>	The returned tuner band. This must be one of the options returned by calls to ASX_Tuner_EnumerateBand() .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.16.2.6 ASX32_API ASX_ERROR ASX_Tuner_GetDabAudioInfo (

ASX_HANDLE *hTuner*, char * *szAudioInfo*, const int *nSize*)

Get a DAB audio information.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>szAudioInfo</i>	a string to return info about the audio service, such as bitrate, mode
<i>nSize</i>	Max size of string

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.7 ASX32 API ASX_ERROR ASX_Tuner_GetDabAudioServiceCount (
ASX_HANDLE hTuner, int * pnIndex, int * pnCount)

Get Number of Dab Audio Services.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>pnIndex</i>	Current index selected
<i>pnCount</i>	Number of available services

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.8 ASX32 API ASX_ERROR ASX_Tuner_GetDabAudioServiceName (
ASX_HANDLE hTuner, char * szAudioServiceName, const int
nSize, const int nIndex)

Get a DAB Audio Service.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>szAudioServiceName</i>	String containing audio service name
<i>nSize</i>	Max size of string
<i>nIndex</i>	Index of service to get

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.9 ASX32 API ASX_ERROR ASX_Tuner_GetDabMultiplexId (
ASX_HANDLE hTuner, unsigned long * dwMultiplexId)

Get a DAB Multiplex ID.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>dwMultiplexId</i>	unsigned long to return Multiplex Id

Returns 0 if there is no error, otherwise one of **asxERROR** is returned.

Get a DAB Multiplex Name.

<i>hTuner</i>	A handle to an ASX tuner control.
<i>szMultiplex-Name</i>	String containing audio service name
<i>nSize</i>	Max size of string

Returns 0 if there is no error, otherwise one of `asxERROR` is returned.

Get a DAB Service ID.

<i>hTuner</i>	A handle to an ASX tuner control.
<i>dwServiceId</i>	unsigned long to return the DAB audio service Id

Returns 0 if there is no error, otherwise one of `asxERROR` is returned.

Get tuner de-emphasis.

<i>hTuner</i>	A handle to an ASX tuner control.
<i>peDeemphasis</i>	The returned de-emphasis value.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.13 ASX32_API ASX_ERROR ASX_Tuner_GetDigitalSignalQuality (
ASX_HANDLE hTuner, int * pnSignalQuality)

Get digital signal quality.

The API supports HDRadio in the USA and DAB elsewhere.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>pnSignalQuality</i>	the returned signal quality between 0(poor)..6(excellent).

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.14 ASX32_API ASX_ERROR ASX_Tuner_GetFirmwareVersion (
ASX_HANDLE hTuner, char * szFirmwareVersion, const int
nStringLength)

Get a Firmware version string.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>szFirmwareVersion</i>	the returned FW Version string
<i>nStringLength</i>	length of string being passed in

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.15 ASX32_API ASX_ERROR ASX_Tuner_GetFrequency (
ASX_HANDLE hTuner, unsigned long * plFreq)

Get the tuner frequency.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>plFreq</i>	The returned frequency.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Note

The tuner frequency may not change immediately. It may take up to 300ms to change. To determine when the frequency has changed, call [ASX_Tuner_GetFrequency\(\)](#) until it returns the new frequency.

Examples:

[tuner/main.c](#).

7.16.2.16 ASX32_API ASX_ERROR ASX_Tuner.GetFrequencyRange (
ASX_HANDLE *hTuner*, const enum asxTUNERBAND
***eBand*, unsigned long * *pMin*, unsigned long * *pMax*, unsigned long * *pStep*)**

Get the tuner frequency range in Hz.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>eBand</i>	Band to get frequency range of.
<i>pMin</i>	The returned minimum frequency in Hz.
<i>pMax</i>	The returned maximum frequency in Hz.
<i>pStep</i>	The returned frequency step in Hz.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.17 ASX32_API ASX_ERROR ASX_Tuner.GetGain (
ASX_HANDLE *hTuner*, float * *pfTunerGain*)

Get the tuner gain.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>pfTunerGain</i>	The returned gain in dB.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.16.2.18 ASX32_API ASX_ERROR ASX_Tuner_GetGainRange (
ASX_HANDLE *hTuner*, float * *fMin*, float * *fMax*, float * *fStep*)

Get the tuner gain range (in dB).

Some tuners controls support a gain adjustment and this control will return the range of gain settings supported.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>fMin</i>	The returned minimum gain in dB.
<i>fMax</i>	The returned maximum gain in dB.
<i>fStep</i>	The returned gain step in dB.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.16.2.19 ASX32_API ASX_ERROR ASX_Tuner_GetHdBlend (
ASX_HANDLE *hTuner*, enum asxTUNERHDBLEND * *pnMode*)

Get a HD Radio tuner analog or digital blend.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>pnMode</i>	0 is automatic switch to digital, 1 is analog only.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.20 ASX32_API ASX_ERROR ASX_Tuner_GetHdRadioDspVersion (
ASX_HANDLE *hTuner*, char * *szSdkVersion*, const int *nStringLength*)

Deprecated

This function has been superseded by [ASX_Tuner_GetFirmwareVersion\(\)](#)

7.16.2.21 ASX32_API ASX_ERROR ASX_Tuner.GetHdRadioSdkVersion (
ASX_HANDLE *hTuner*, char * *szSdkVersion*, const int
***nStringLength*)**

Deprecated

This function has been superseded by [ASX_Tuner_GetFirmwareVersion\(\)](#)

7.16.2.22 ASX32_API ASX_ERROR ASX_Tuner.GetHdRadioSignalQuality (
ASX_HANDLE *hTuner*, int * *pnSignalQuality*)

Deprecated

This function has been superseded by [ASX_Tuner_GetDigitalSignalQuality\(\)](#)

7.16.2.23 ASX32_API ASX_ERROR ASX_Tuner.GetMode (
ASX_HANDLE *hTuner*, const enum asxTUNERMODE
***eMode*, enum asxTUNERMODE * *peSetting*)**

Gets the tuner mode.

Currently this can only be used for turning the RSS level reading on and off on an MT4039 tuner.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>eMode</i>	The only valid parameter for this call is asxTUNERMODE_RSS .
<i>peSetting</i>	Returns the mode setting. Only current valid values are asxTUNERMODE_RSS_ENABLE or asxTUNERMODE_RSS_DISABLE .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.24 ASX32_API ASX_ERROR ASX_Tuner.GetProgram (
ASX_HANDLE *hTuner*, enum asxTUNERPROGRAM *
***peProgram*)**

Get tuner program.

The API only supports HDRadio in the USA.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>peProgram</i>	The returned program setting.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.25 ASX32_API ASX_ERROR ASX_Tuner_GetRawRFLevel (
ASX_HANDLE hTuner, int * nRawRFLevel)

Get the Raw tuner RF level.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>nRawRFLevel</i>	The returned Raw RF level in whatever units the tuner supports.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.26 ASX32_API ASX_ERROR ASX_Tuner_GetRFLevel (
ASX_HANDLE hTuner, float * nRFLevel)

Get the tuner RF level.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>nRFLevel</i>	The returned RF level in dBuV.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.16.2.27 ASX32_API ASX_ERROR ASX_Tuner_GetStatus (
ASX_HANDLE hTuner, unsigned int * puErrorStatusMask,
unsigned int * puErrorStatus)

Get the tuner status.

This function gets the tuner status and indicates which bits are valid for the current status reading.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
---------------	-----------------------------------

<i>puErrorStatusMask</i>	The returned status mask. This mask indicates which status bits are valid in <i>puErrorStatus</i> . Bits are defined by asxTUNER_STATUS .
<i>puErrorStatus</i>	The returned status bits.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.16.2.28 ASX32_API ASX_ERROR ASX_Tuner.SetBand (
ASX_HANDLE *hTuner*, const enum asxTUNERBAND *eBand*
)

Set the tuner band.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>eBand</i>	The tuner band to set. This must be one of the options returned by calls to ASX_Tuner_EnumerateBand() .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.16.2.29 ASX32_API ASX_ERROR ASX_Tuner.SetDabAudioService (
ASX_HANDLE *hTuner*, const int *nIndex*)

Set a DAB Audio Service.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>nIndex</i>	Index of service to tune to

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.16.2.33 ASX32_API ASX_ERROR ASX_Tuner_SetHdBlend (
ASX_HANDLE *hTuner*, const enum asxTUNERHDBLEND
***nMode*)**

Set a HD Radio tuner to analog only or auto switch.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>nMode</i>	0 is automatic switch to digital, 1 is analog only.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.34 ASX32_API ASX_ERROR ASX_Tuner_SetMode (
ASX_HANDLE *hTuner*, const enum asxTUNERMODE
***eMode*, const enum asxTUNERMODE *eSetting*)**

Sets the tuner mode.

Currently this can only be used for turning the RSS level reading on and off on an MT4039 tuner.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>eMode</i>	The only valid parameter for this call is asxTUNERMODE_RSS .
<i>eSetting</i>	The mode setting. Only current valid values are asxTUNERMODE_RSS_ENABLE or asxTUNERMODE_RSS_DISABLE .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.16.2.35 ASX32_API ASX_ERROR ASX_Tuner_SetProgram (
ASX_HANDLE *hTuner*, const enum asxTUNERPROGRAM
***eProgram*)**

Set tuner program.

To The API only supports HDRadio in the USA.

Parameters

<i>hTuner</i>	A handle to an ASX tuner control.
<i>eProgram</i>	The program to set.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.17 PAD control functions

These functions support Program Auxiliary Data for HD Radio and RDS for FM analog.

Functions

- [ASX_PAD_GetChannelName](#)
Get a Program Auxiliary Data channel name.
- [ASX_PAD_GetArtist](#)
Get a Program Auxiliary Data artist.
- [ASX_PAD_GetTitle](#)
Get a Program Auxiliary Data title.
- [ASX_PAD_GetComment](#)
Get a Program Auxiliary Data comment.
- [ASX_PAD_GetProgramType](#)
Get a Program Auxiliary Data program type (PTY).
- [ASX_PAD_GetProgramTypeString](#)
Get a Program Auxiliary Data PTY string.
- [ASX_PAD_GetRdsPI](#)
Get a Program Identification number.

7.17.1 Detailed Description

These functions support Program Auxiliary Data for HD Radio and RDS for FM analog.

7.17.2 Function Documentation

7.17.2.1 ASX32 API `ASX_ERROR ASX_PAD.GetArtist ( ASX_HANDLE hPAD, char * pszArtist, const int nStringLength )`

Get a Program Auxiliary Data artist.

This control will only ever exist on a tuner node.

Parameters

<i>hPAD</i>	A handle to an ASX PAD control.
<i>pszArtist</i>	The string buffer of size <i>nFieldLength</i> allocated by the caller.
<i>nStringLength</i>	The length of the <i>pszArtist</i> buffer. Should be ASX_LONG_STRING.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.17.2.2 ASX32_API ASX_ERROR ASX_PAD_GetChannelName (
ASX_HANDLE hPAD, char * pszChannelName, const int
nStringLength)

Get a Program Auxiliary Data channel name.

This control will only ever exist on a tuner node.

Parameters

<i>hPAD</i>	A handle to an ASX PAD control.
<i>pszChannel-Name</i>	The string buffer of size <i>nFieldLength</i> allocated by the caller.
<i>nStringLength</i>	The length of the <i>pszChannelName</i> buffer. Should be ASX_SHORT_STRING.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.17.2.3 ASX32_API ASX_ERROR ASX_PAD_GetComment (
ASX_HANDLE hPAD, char * pszComment, const int
nStringLength)

Get a Program Auxiliary Data comment.

This control will only ever exist on a tuner node.

Parameters

<i>hPAD</i>	A handle to an ASX PAD control.
<i>pszComment</i>	The string buffer of size <i>nStringLength</i> allocated by the caller.
<i>nStringLength</i>	The length of the <i>pszComment</i> buffer. Should be ASX_LONGLONG_STRING.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.17.2.4 ASX32 API ASX_ERROR ASX_PAD.GetProgramType (ASX_HANDLE hPAD, int * pnProgramType)

Get a Program Auxiliary Data program type (PTY).

This control will only ever exist on a tuner node.

Parameters

<i>hPAD</i>	A handle to an ASX PAD control.
<i>pnProgram-Type</i>	Returns the program type (PTY).

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.17.2.5 ASX32 API ASX_ERROR ASX_PAD.GetProgramTypeString (ASX_HANDLE hPAD, const enum asxTUNER_RDS_TYPE eType, const int nPTY, char * pszString, const int nStringLength)

Get a Program Auxiliary Data PTY string.

This control will only ever exist on a tuner node.

Parameters

<i>hPAD</i>	A handle to an ASX PAD control.
<i>eType</i>	The RDS/RBDS type selection.
<i>nPTY</i>	The program type code to translate.
<i>pszString</i>	The string buffer of size <i>nStringLength</i> allocated by the caller.
<i>nStringLength</i>	The length of the <i>pszComment</i> buffer. Should be ASX_LONGLONG_STRING.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.17.2.6 ASX32_API ASX_ERROR ASX_PAD_GetRdsPI (
ASX_HANDLE hPAD, int * uPI)

Get a Program Identification number.

This control will only ever exist on a tuner node. A valid PI will only be returned for a tuner set to analog FM.

Parameters

<i>hPAD</i>	A handle to an ASX PAD control.
<i>uPI</i>	The returned program identification number.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.17.2.7 ASX32_API ASX_ERROR ASX_PAD_GetTitle (
ASX_HANDLE hPAD, char * pszTitle, const int nStringLength)

Get a Program Auxiliary Data title.

This control will only ever exist on a tuner node.

Parameters

<i>hPAD</i>	A handle to an ASX PAD control.
<i>pszTitle</i>	The string buffer of size nFieldLength allocated by the caller.
<i>nStringLength</i>	The length of the pszTitle buffer. Should be ASX_LONG_STRING.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[tuner/main.c](#).

7.18 Sample clock control functions

These functions implement sample clock operations that manipulate the adapter's sample clock generation.

Functions

- [ASX_SampleClock_EnumerateSampleRate](#)
Enumerate each sample clock rates for the local sample clock generator.
- [ASX_SampleClock_EnumerateLocalRate](#)
Enumerate each sample clock rates for the local sample clock generator.
- [ASX_SampleClock_SetSampleRate](#)
Set the sample rate for the local sample clock generator.
- [ASX_SampleClock_SetLocalRate](#)
Set the sample rate for the local sample clock generator.
- [ASX_SampleClock_GetSampleRate](#)
Get the adapter's sample rate.
- [ASX_SampleClock_GetLocalRate](#)
Get the sample rate for the local sample clock generator.
- [ASX_SampleClock_EnumerateClockSource](#)
Enumerate each sample clock source option.
- [ASX_SampleClock_SetClockSource](#)
Set the sample clock source.
- [ASX_SampleClock_GetClockSource](#)
Get the sample clock source.
- [ASX_SampleClock_SetAutoSource](#)
Set the sample clock to automatically source its clock from a valid input.
- [ASX_SampleClock_GetAutoSource](#)
Get the setting of the auto source property of the sample clock.
- [ASX_SampleClock_SetLocalRateLock](#)
Lock the local sample clock to its current setting.
- [ASX_SampleClock_GetLocalRateLock](#)
Get the setting of the local sample clock lock.

7.18.1 Detailed Description

These functions implement sample clock operations that manipulate the adapter's sample clock generation.

7.18.2 Function Documentation

7.18.2.1 ASX32_API ASX_ERROR ASX_SampleClock_EnumerateClockSource (
ASX_HANDLE *hSampleClock*, const int *nIndex*, enum
asxSAMPLE_CLOCK_SOURCE * *peClockSource*, int * *pnCount*)

Enumerate each sample clock source option.

Parameters

<i>hSample-Clock</i>	A handle to an ASX sample clock control.
<i>nIndex</i>	The index of the sample clock source option to fetch.
<i>peClock-Source</i>	The returned sample clock source option.
<i>pnCount</i>	The total number of available sample clock sources.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.18.2.2 ASX32_API ASX_ERROR ASX_SampleClock_EnumerateLocalRate (
ASX_HANDLE *hSampleClock*, const int *nIndex*, enum
asxSAMPLE_RATE * *peSampleRate*, int * *pnCount*)

Enumerate each sample clock rates for the local sample clock generator.

Parameters

<i>hSample-Clock</i>	A handle to an ASX sample clock control.
<i>nIndex</i>	The index of the sample clock rate option to fetch.
<i>peSampleRate</i>	The returned sample rate option.
<i>pnCount</i>	The total number of available sample rates.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.18.2.3 ASX32_API ASX_ERROR ASX_SampleClock_EnumerateSampleRate (
ASX_HANDLE *hSampleClock*, const int *nIndex*, enum
asxSAMPLE_RATE * *peSampleRate*, int * *pnCount*)

Deprecated

This function has been superseded by [ASX_SampleClock_EnumerateLocalRate\(\)](#)

7.18.2.4 ASX32 API ASX_ERROR ASX_SampleClock_GetAutoSource (
ASX_HANDLE *hSampleClock*, int * *pnEnable*)

Get the setting of the auto source property of the sample clock.

Parameters

<i>hSample-Clock</i>	A handle to an ASX sample clock control.
<i>*pnEnable</i>	Returns 1 if enabled or 0 if not.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.18.2.5 ASX32 API ASX_ERROR ASX_SampleClock_GetClockSource (
ASX_HANDLE *hSampleClock*, enum
asxSAMPLE_CLOCK_SOURCE * *peClockSource*)

Get the sample clock source.

Parameters

<i>hSample-Clock</i>	A handle to an ASX sample clock control.
<i>peClock-Source</i>	The returned sample clock source.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.18.2.6 ASX32 API ASX_ERROR ASX_SampleClock_GetLocalRate (
ASX_HANDLE *hSampleClock*, int * *pnSampleRate*)

Get the sample rate for the local sample clock generator.

Parameters

<i>hSample-Clock</i>	A handle to an ASX sample clock control.
<i>pnSampleRate</i>	The returned sample rate.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.18.2.7 ASX32_API ASX_ERROR ASX_SampleClock_GetLocalRateLock (ASX_HANDLE *hSampleClock*, int * *pnLock*)

Get the setting of the local sample clock lock.

Parameters

<i>hSample-Clock</i>	A handle to an ASX sample clock control.
<i>*pnLock</i>	Returns 1 if enabled or 0 if not.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.18.2.8 ASX32_API ASX_ERROR ASX_SampleClock_GetSampleRate (ASX_HANDLE *hSampleClock*, int * *pnSampleRate*)

Get the adapter's sample rate.

Parameters

<i>hSample-Clock</i>	A handle to an ASX sample clock control.
<i>pnSampleRate</i>	The returned sample rate.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.18.2.9 ASX32_API ASX_ERROR ASX_SampleClock_SetAutoSource (ASX_HANDLE *hSampleClock*, const int *nEnable*)

Set the sample clock to automatically source its clock from a valid input.

Parameters

<i>hSample-Clock</i>	A handle to an ASX sample clock control.
<i>nEnable</i>	When set to 1 the auto source is enabled, when 0 it is disabled.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.18.2.13 ASX32_API ASX_ERROR ASX_SampleClock_SetSampleRate (
ASX_HANDLE hSampleClock, const int nSampleRate)

Deprecated

This function has been superseded by [ASX_SampleClock_SetLocalRate\(\)](#)

7.19 AESEBU receiver control functions

These functions implement AESEBU receiver operations.

Functions

- [ASX_AESEBUReceiver_GetErrorStatus](#)
Get the status of the AESEBU receiver.
- [ASX_AESEBUReceiver_GetSampleRate](#)
Get the sample rate of the AESEBU receiver.
- [ASX_AESEBUReceiver_EnumerateFormat](#)
Enumerate each AES3 receive format supported by the hardware.
- [ASX_AESEBUReceiver_SetFormat](#)
Set the format of the AESEBU receiver.
- [ASX_AESEBUReceiver_GetFormat](#)
Get the format of the AESEBU receiver.

7.19.1 Detailed Description

These functions implement AESEBU receiver operations.

7.19.2 Function Documentation

7.19.2.1 ASX32_API ASX_ERROR ASX_AESEBUReceiver_EnumerateFormat (
ASX_HANDLE hAESEBURx, const int nIndex, enum
asxAESEBU_FORMAT *peAesebuFormat, int *pnCount)

Enumerate each AES3 receive format supported by the hardware.

Parameters

<i>hAESE-BURx</i>	A handle to an ASX AESEBU receiver control.
<i>nIndex</i>	The index of the format option to fetch.
<i>peAesebu-Format</i>	The returned format option.
<i>pnCount</i>	The total number of available formats.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.19.2.2 ASX32 API ASX_ERROR ASX_AESEBUReceiver_GetErrorStatus (ASX_HANDLE hAESEBURx, unsigned int * pdwErrorStatusMask, unsigned int * pdwErrorStatus)

Get the status of the AESEBU receiver.

Parameters

<i>hAESE-BURx</i>	A handle to an ASX AESEBU receiver control.
<i>pdwErrorStatus-Mask</i>	A bit mask field indicating which of the bitfields defined by asxAESEBU_STATUS are returned.
<i>pdwErrorStatus</i>	The returned status fields.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.19.2.3 ASX32 API ASX_ERROR ASX_AESEBUReceiver_GetFormat (ASX_HANDLE hAESEBURx, enum asxAESEBU_FORMAT * peAesebuFormat)

Get the format of the AESEBU receiver.

Parameters

<i>hAESE-BURx</i>	A handle to an ASX AESEBU receiver control.
<i>peAesebu-Format</i>	The returned format.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.19.2.4 ASX32_API ASX_ERROR ASX_AESEBUReceiver_GetSampleRate (ASX_HANDLE hAESEBURx, unsigned int * pdwSampleRate)

Get the sample rate of the AESEBU receiver.

Parameters

<i>hAESEBURx</i>	A handle to an ASX AESEBU receiver control.
<i>pdwSampleRate</i>	The returned sample rate.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.19.2.5 ASX32_API ASX_ERROR ASX_AESEBUReceiver_SetFormat (ASX_HANDLE hAESEBURx, const enum asxAESEBU_FORMAT eAesebuFormat)

Set the format of the AESEBU receiver.

Parameters

<i>hAESEBURx</i>	A handle to an ASX AESEBU receiver control.
<i>eAesebuFormat</i>	The mode to set the receiver control to.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.20 AESEBU transmitter control functions

These functions implement AESEBU transmitter operations.

Functions

- [ASX_AESEBUTransmitter_EnumerateFormat](#)
Enumerate each AES3 transmit format supported by the hardware.
- [ASX_AESEBUTransmitter_SetFormat](#)
Set the format of the AESEBU transmitter.
- [ASX_AESEBUTransmitter_GetFormat](#)
Get the format of the AESEBU transmitter.

7.20.1 Detailed Description

These functions implement AESEBU transmitter operations.

7.20.2 Function Documentation

7.20.2.1 ASX32_API ASX_ERROR ASX_AESEBUTransmitter_EnumerateFormat (
ASX_HANDLE hAESEBUTx, const int nIndex, enum
asxAESEBU_FORMAT *peAesebuFormat, int *pnCount)

Enumerate each AES3 transmit format supported by the hardware.

Parameters

<i>hAESEBUTx</i>	A handle to an ASX AESEBU transmitter control.
<i>nIndex</i>	The index of the format option to fetch.
<i>peAesebu-Format</i>	The returned format option.
<i>pnCount</i>	The total number of available formats.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.20.2.2 ASX32_API ASX_ERROR ASX_AESEBUTransmitter_GetFormat (
ASX_HANDLE hAESEBUTx, enum asxAESEBU_FORMAT
*** peAesebuFormat)**

Get the format of the AESEBU transmitter.

Parameters

<i>hAESEBUTx</i>	A handle to an ASX AESEBU transmitter control.
<i>peAesebu-Format</i>	The returned format.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.20.2.3 ASX32_API ASX_ERROR ASX_AESEBUTransmitter_SetFormat (
ASX_HANDLE hAESEBUTx, const enum
asxAESEBU_FORMAT eAesebuFormat)

Set the format of the AESEBU transmitter.

Parameters

<i>hAESEBUTx</i>	A handle to an ASX AESEBU transmitter control.
<i>eAesebuFormat</i>	The format to set the transmitter control to.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.21 GPIO control functions

These functions implement GPIO operations.

Functions

- [ASX_GPIO_GetProperties](#)
Get the properties of the GPIO control.
- [ASX_GPIO_InputGet](#)
Read the state of the GPIO opto inputs.
- [ASX_GPIO_OutputSet](#)
Write to the GPIO relay outputs.
- [ASX_GPIO_OutputGet](#)
Read the current GPIO relay output settings.

7.21.1 Detailed Description

These functions implement GPIO operations. By GPIO we here mean the control of relays and the reading of optos.

7.21.2 Function Documentation

7.21.2.1 ASX32_API ASX_ERROR ASX_GPIO_GetProperties (
ASX_HANDLE hGPIO, int * pnNumberOfInputBits, int *
pnNumberOfOutputBits)

Get the properties of the GPIO control.

GPIO controls have a number of input and output bits and this function tells the application how many there are of each.

Parameters

<i>hGPIO</i>	A handle to an ASX GPIO control object.
<i>pnNumberOfInputBits</i>	The number of input bits.
<i>pnNumberOfOutputBits</i>	The number of output bits.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.21.2.2 ASX32 API ASX_ERROR ASX_GPIO_InputGet (
ASX_HANDLE hGPIO, int * pnInputBits, const int
nNumberOfBits)

Read the state of the GPIO opto inputs.

This functions reads all the GPIO inputs and returns the current readings in an array of bits.

Parameters

<i>hGPIO</i>	A handle to an ASX GPIO control object.
<i>pnInputBits</i>	An array of dimension nNumberOfBits items elements returns the bit readings. pnBits[0] is the reading of opto 0 and pnBits[1] is the reading of opto 1. All pnBits elements will be set to either 1 or 0.
<i>nNumberOfBits</i>	The number of bits which should be set to the dimension of pnBits.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.21.2.3 ASX32 API ASX_ERROR ASX_GPIO_OutputGet (
ASX_HANDLE hGPIO, int * pnOutputBits, const int
nNumberOfBits)

Read the current GPIO relay output settings.

This functions reads all the GPIO outputs.

Parameters

<i>hGPIO</i>	A handle to an ASX GPIO control object.
<i>pnOutputBits</i>	An array of dimension nNumberOfBits elements that is used to return the relay settings.
<i>nNumberOfBits</i>	The number of bits which must be set to the dimension of pnBits.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.21.2.4 ASX32_API ASX_ERROR ASX_GPIO.OutputSet (
ASX_HANDLE hGPIO, int * pnOutputBits, const int
nNumberOfBits)

Write to the GPIO relay outputs.

This functions writes to all the GPIO outputs, setting the outputs to the values of pnBits.

Parameters

<i>hGPIO</i>	A handle to an ASX GPIO control object.
<i>pnOutput-Bits</i>	An array of dimension nNumberOfBits elements that is used to set the relays. All elements should be set to a value of either 1 or 0.
<i>nNumberOf-Bits</i>	The number of bits which must be set to the dimension of pnBits.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.22 Vox control functions

These functions support vox control manipulation.

Functions

- [ASX_Vox_SetLevel](#)
Set vox level.
- [ASX_Vox_GetLevel](#)
Get vox level.
- [ASX_Vox_GetRange](#)
Get that range of vox settings available.

7.22.1 Detailed Description

These functions support vox control manipulation.

7.22.2 Function Documentation

7.22.2.1 ASX32_API ASX_ERROR ASX_Vox_GetLevel (ASX_HANDLE *hVox*, float * *fGetLevel*)

Get vox level.

Parameters

<i>hVox</i>	A handle to an ASX vox control.
<i>fGetLevel</i>	The returned gain in dBFS, i.e. 0dB is fullscale.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.22.2.2 ASX32_API ASX_ERROR ASX_Vox_GetRange (ASX_HANDLE *hVox*, float * *fMinLevel*, float * *fMaxLevel*, float * *fLevelStep*)

Get that range of vox settings available.

Parameters

<i>hVox</i>	A handle to an ASX vox control.
<i>fMinLevel</i>	The returned minimum gain in dBFS, i.e. 0dB is fullscale.
<i>fMaxLevel</i>	The returned maximum gain in dBFS, i.e. 0dB is fullscale.
<i>fLevelStep</i>	The returned gain stepsize in dB.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.22.2.3 ASX32_API ASX_ERROR ASX_Vox_SetLevel (ASX_HANDLE *hVox*, const float *fSetLevel*)

Set vox level.

Parameters

<i>hVox</i>	A handle to an ASX vox control.
<i>fSetLevel</i>	The gain to set in dBFS, i.e. 0dB is fullscale.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.23 Generic control functions

This function supports the generic control.

Functions

- [ASX_GetGenericControlName](#)

Get the name of the control.

7.23.1 Detailed Description

This function supports the generic control.

7.23.2 Function Documentation

7.23.2.1 ASX32_API ASX_ERROR ASX_GetGenericControlName (
ASX_HANDLE *hControl*, char * *pName*)

Get the name of the control.

Parameters

<i>hControl</i>	A handle to an ASX generic control.
<i>pName</i>	Pointer to a buffer (at least 60 chars) for the name.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.24 Microphone control functions

These functions implement Microphone phantom power setting.

Functions

- [ASX_Mic_SetPhantomPower](#)

Turn the phantom power on or off.

- [ASX_Mic_GetPhantomPower](#)

Get the current state of the phantom power (on or off).

7.24.1 Detailed Description

These functions implement Microphone phantom power setting.

7.24.2 Function Documentation

7.24.2.1 ASX32_API ASX_ERROR ASX_Mic_GetPhantomPower (ASX_HANDLE hMic, int * pOnOff)

Get the current state of the phantom power (on or off).

Parameters

<i>hMic</i>	A handle to an ASX Microphone control object.
<i>pOnOff</i>	Receives power state 1 = Phantom power is on. 0 = Phantom power is off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.24.2.2 ASX32_API ASX_ERROR ASX_Mic_SetPhantomPower (ASX_HANDLE hMic, const int nOnOff)

Turn the phantom power on or off.

Parameters

<i>hMic</i>	A handle to an ASX Microphone control object.
<i>nOnOff</i>	1 = Turn phantom power on. 0 = Turn phantom power off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.25 Parametric Equalizer control functions

These functions implement Parametric Equalizer settings.

Functions

- [ASX_EQ_GetInfo](#)
Gets information on the equalizer.
- [ASX_EQ_SetState](#)
Turns the equalizer on or off.

- [ASX_EQ_SetBand](#)

Sets the parameters for an equalizer band.

- [ASX_EQ_GetBand](#)

Gets the parameters for an equalizer band.

7.25.1 Detailed Description

These functions implement Parametric Equalizer settings.

7.25.2 Function Documentation

7.25.2.1 ASX32_API ASX_ERROR ASX_EQ_GetBand (
ASX_HANDLE hParmEq, const unsigned short wIndex, enum
asxEQBANDTYPE * peType, unsigned long * pdwFrequencyHz, short * pnQ100,
short * pnGain0_01dB)

Gets the parameters for an equalizer band.

Parameters

<i>hParmEq</i>	A handle to an ASX Parametric Equalizer control object.
<i>wIndex</i>	Zero based index of the band, should be between zero and nBands - 1, where nBands is the number of bands from a call to ASX_EQ_GetInfo() .
<i>peType</i>	Receives the type of the band.
<i>pdwFrequencyHz</i>	Receives the cutoff (for high/lo types) or center frequency (for band types).
<i>pnQ100</i>	Receives the sharpness (for high/lo types) or bandwidth (for band types). Value can be positive or negative and is given in hundreths (nQ100 = 72 means Q = 0.72).
<i>pnGain0_01dB</i>	Receives the gain for equalizer, high shelf and low shelf filter types. Value can be positive or negative (0 is unity gain) and is given in hundreths.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.25.2.2 ASX32_API ASX_ERROR ASX_EQ_GetInfo (
ASX_HANDLE hParmEq, unsigned short * pwNumberOfBands,
unsigned short * pwEnabled)

Gets information on the equalizer.

Parameters

<i>hParmEq</i>	A handle to an ASX Parametric Equalizer control object.
----------------	---

<i>pwNumberOfBands</i>	Receives the number of bands.
<i>pwEnabled</i>	Receives the on/off state of the equalizer.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.25.2.3 ASX32 API ASX_ERROR ASX_EQ_SetBand (
ASX_HANDLE hParmEq, const unsigned short wIndex, const
enum asxEQBANDTYPE eType, const unsigned long dwFrequencyHz, const
short nQ100, const short nGain0_01dB)

Sets the parameters for an equalizer band.

Parameters

<i>hParmEq</i>	A handle to an ASX Parametric Equalizer control object.
<i>wIndex</i>	Zero based index of the band, should be between zero and nBands - 1. where nBands is the number of bands from a call to ASX_EQ_GetInfo() .
<i>eType</i>	Specifies the effect of the band.
<i>dwFrequencyHz</i>	Specifies the cutoff (for high/lo types) or center frequency (for band types).
<i>nQ100</i>	Specifies the sharpness (for high/lo types) or bandwidth (for band types). Value can be positive or negative and is given in hundredths (nQ100 = 72 means Q = 0.72).
<i>nGain0_01dB</i>	Specifies the gain for equalizer, high shelf and low shelf filter types. Value can be positive or negative (0 is unity gain) and is given in hundredths.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.25.2.4 ASX32 API ASX_ERROR ASX_EQ_SetState (
ASX_HANDLE hParmEq, const unsigned short wOnOff)

Turns the equalizer on or off.

Parameters

<i>hParmEq</i>	A handle to an ASX Parametric Equalizer control object.
<i>wOnOff</i>	1 = Turn equalizer on. 0 = Turn equalizer off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26 Compander control functions

These functions implement Compander settings.

Functions

- [ASX_Compander_Set](#)
- [ASX_Compander_Get](#)
- [ASX_Compander_SetEnable](#)
Sets the on/off parameter for the compander.
- [ASX_Compander_GetEnable](#)
Gets the on/off parameter for the compander.
- [ASX_Compander_SetMakeupGain](#)
Set the compander makeup gain.
- [ASX_Compander_GetMakeupGain](#)
Get the compander makeup gain.
- [ASX_Compander_SetAttackTimeConstant](#)
Set the attack time constant in ms.
- [ASX_Compander_GetAttackTimeConstant](#)
Get the attack time constant in ms.
- [ASX_Compander_SetDecayTimeConstant](#)
Set the decay time constant in ms.
- [ASX_Compander_GetDecayTimeConstant](#)
Get the decay time constant in ms.
- [ASX_Compander_SetThreshold](#)
Set the compander threshold in dbFS.
- [ASX_Compander_GetThreshold](#)
Get the compander threshold in dbFS.
- [ASX_Compander_SetRatio](#)
Set the compander ratio (slope).
- [ASX_Compander_GetRatio](#)
Get the compander ratio (slope).

7.26.1 Detailed Description

These functions implement Comander settings.

7.26.2 Function Documentation

7.26.2.1 ASX32_API ASX_ERROR ASX_Comander_Get (
ASX_HANDLE hComander, unsigned short * pwAttack,
unsigned short * pwDecay, short * pwRatio100, short * pnThreshold0_01dB, short * pnMakeupGain0_01dB)

Deprecated

This function has been broekn up in to individual Get() functions. Gets the parameters for the comander.

Parameters

<i>hComander</i>	A handle to an ASX Comander control object.
<i>pwAttack</i>	Gets comander attack time in milliseconds.
<i>pwDecay</i>	Gets comander decay time in milliseconds.
<i>pwRatio100</i>	Gets the input to output gain ratio. Value is given in hundreths (wRatio100 = 72 means 0.72).
<i>pnThreshold0_01dB</i>	Gets the threshold above which the ratio applies. Value is given in hundreths.
<i>pnMakeupGain0_01dB</i>	Receives the positive or negative offset to the output gain. Value is given in hundreths.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.2 ASX32_API ASX_ERROR ASX_Comander_GetAttackTimeConstant (
ASX_HANDLE hComander, enum
asxCOMPANDER_INDEX index, unsigned int * pnAttack)

Get the attack time constant in ms.

Parameters

<i>hComander</i>	A handle to an ASX Comander control object.
<i>index</i>	Index of timeconstant to get.
<i>pnAttack</i>	Gets attack time in milliseconds.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.3 ASX32_API ASX_ERROR ASX_Compander_GetDecayTimeConstant (
ASX_HANDLE *hCompander*, enum
asxCOMPANDER_INDEX *index*, unsigned int * *pnDecay*)

Get the decay time constant in ms.

Parameters

<i>hCompander</i>	A handle to an ASX Compander control object.
<i>index</i>	Index of timeconstant to get.
<i>pnAttack</i>	Gets decay time in milliseconds.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.4 ASX32_API ASX_ERROR ASX_Compander_GetEnable (
ASX_HANDLE *hCompander*, unsigned int * *nOn*)

Gets the on/off parameter for the compander.

Parameters

<i>hCompander</i>	A handle to an ASX Compander control object.
<i>nOn</i>	Gets the compander ton/off status.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.5 ASX32_API ASX_ERROR ASX_Compander_GetMakeupGain (
ASX_HANDLE *hCompander*, float * *fMakeupGain*)

Get the compander makeup gain.

Parameters

<i>hCompander</i>	A handle to an ASX Compander control object.
<i>fMakeupGain</i>	Returns the current makeup gain in dBFS.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.6 ASX32_API ASX_ERROR ASX_Compander_GetRatio (
 ASX_HANDLE hCompander, enum
 asxCOMPANDER_INDEX index, float * fRatio)

Get the compander ratio (slope).

Parameters

<i>hCompan- der</i>	A handle to an ASX Compander control object.
<i>index</i>	Index of rtio to get.
<i>fRatio</i>	Returned ratio. 1.0 is 1:1.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.7 ASX32_API ASX_ERROR ASX_Compander_GetThreshold (
 ASX_HANDLE hCompander, enum
 asxCOMPANDER_INDEX index, float * pnThreshold)

Get the compander threshold in dbFS.

Parameters

<i>hCompan- der</i>	A handle to an ASX Compander control object.
<i>index</i>	Index of threshold to get.
<i>pnThreshold</i>	Returned threshold in dbFS.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.8 ASX32_API ASX_ERROR ASX_Compander_Set (
 ASX_HANDLE hCompander, const unsigned short wAttack,
 const unsigned short wDecay, const short wRatio100, const short nThreshold0_01dB,
 const short nMakeupGain0_01dB)

Deprecated

This function has been broekn up in to individual Set() functions. Sets the parameters for the compander.

Parameters

<i>hCompan- der</i>	A handle to an ASX Compander control object.
<i>wAttack</i>	Sets compander attack time in milliseconds.
<i>wDecay</i>	Sets compander decay time in milliseconds.
<i>wRatio100</i>	Sets the input to output gain ratio. Value is given in hundreths (<i>wRatio100</i> = 72 means 0.72).
<i>nThreshold0_ 01dB</i>	Sets the threshold above which the ratio applies. Value is given in hundreths.
<i>nMakeupGain_ 01dB</i>	Adds a positive or negative offset to the output gain. Value is given in hundreths.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.9 ASX32_API ASX_ERROR ASX_Compander.SetAttackTimeConstant (
ASX_HANDLE hCompan-der, enum
asxCOMPANDER_INDEX index, const unsigned int nAttack)

Set the attack time constant in ms.

Parameters

<i>hCompan- der</i>	A handle to an ASX Compander control object.
<i>index</i>	Index of timeconstant to set.
<i>nAttack</i>	Sets attack time in milliseconds.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.10 ASX32_API ASX_ERROR ASX_Compander.SetDecayTimeConstant (
ASX_HANDLE hCompan-der, enum
asxCOMPANDER_INDEX index, const unsigned int nDecay)

Set the decay time constant in ms.

Parameters

<i>hCompan- der</i>	A handle to an ASX Compander control object.
<i>index</i>	Index of timeconstant to set.
<i>nDecay</i>	Sets decay time in milliseconds.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.11 ASX32_API ASX_ERROR ASX_Compander.SetEnable (
ASX_HANDLE *hCompander*, const unsigned int *nOn*)

Sets the on/off parameter for the compander.

Parameters

<i>hCompander</i>	A handle to an ASX Compander control object.
<i>nOn</i>	Sets compander to on if set to 1.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.12 ASX32_API ASX_ERROR ASX_Compander.SetMakeupGain (
ASX_HANDLE *hCompander*, const float *fMakeupGain*)

Set the compander makeup gain.

Parameters

<i>hCompander</i>	A handle to an ASX Compander control object.
<i>fMakeupGain</i>	The makeup gain to set in dBFS.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.13 ASX32_API ASX_ERROR ASX_Compander.SetRatio (
ASX_HANDLE *hCompander*, enum
asxCOMPANDER_INDEX *index*, const float *fRatio*)

Set the compander ratio (slope).

Parameters

<i>hCompander</i>	A handle to an ASX Compander control object.
<i>index</i>	Index of ratio to set.
<i>fRatio</i>	Ratio to set. 1.0 is 1:1.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.26.2.14 ASX32_API ASX_ERROR ASX_Compander_SetThreshold (
ASX_HANDLE hCompander, enum
asxCOMPANDER_INDEX index, const float nThreshold)

Set the compander threshold in dbFS.

Parameters

<i>hCompander</i>	A handle to an ASX Compander control object.
<i>index</i>	Index of threshold to set.
<i>nThreshold</i>	Threshold in dbFS.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27 CobraNet control functions

These functions implement device wide CobraNet settings.

Functions

- [ASX_Cobranet_EnumerateModes](#)
- [ASX_Cobranet_GetMode](#)
- [ASX_Cobranet_SetMode](#)
- [ASX_Cobranet_GetIPAddress](#)
Get the current IP address of the Cobranet device.
- [ASX_Cobranet_SetIPAddress](#)
Set the current IP address of the Cobranet device.
- [ASX_Cobranet_GetStaticIPAddress](#)
Get the static IP address of the Cobranet device.
- [ASX_Cobranet_SetStaticIPAddress](#)
Set the static IP address of the Cobranet device.
- [ASX_Cobranet_GetMACAddress](#)
Get the current cobranet MAC address.
- [ASX_Cobranet_GetDescription](#)

Get the device's description from the sysDescr SNMP field.

- [ASX_Cobranet_GetName](#)

Get the device's name from the sysName SNMP field.

- [ASX_Cobranet_SetName](#)

Set the device's name in the sysName SNMP field.

- [ASX_Cobranet_GetLocation](#)

Get the device's location from the sysLocation SNMP field.

- [ASX_Cobranet_SetLocation](#)

Set the device's location in the sysLocation SNMP field.

- [ASX_Cobranet_GetFirmwareRevision](#)

Gets a device's firmware revision.

- [ASX_Cobranet_GetErrorInfo](#)

Gets a device's error information.

- [ASX_Cobranet_GetLatencyAndSampleRate](#)

Gets a device's latency and sample reate.

- [ASX_Cobranet_SetLatencyAndSampleRate](#)

Gets a device's latency and sample reate.

- [ASX_Cobranet_GetPersistence](#)

Gets a device's flash persistence setting.

- [ASX_Cobranet_SetPersistence](#)

Sets a device's flash persistence state.

- [ASX_Cobranet_GetConductorPriority](#)

Gets a device's conductor priority.

- [ASX_Cobranet_SetConductorPriority](#)

Sets a device's conductor priority.

- [ASX_Cobranet_GetConductorStatus](#)

Gets a device's conductor status.

- [ASX_Cobranet_SetSerialEnable](#)

Enable or disable a device's serial bridge.

- [ASX_Cobranet_GetSerialEnable](#)

Gets a device's serial bridge status.

- [ASX_Cobranet_SetSerialConfig](#)
Configures a device's serial bridge.
- [ASX_Cobranet_GetSerialConfig](#)
Gets a device's serial bridge configuration.
- [ASX_Cobranet_GetIfStatus](#)
Gets a device's ethernet connection status.

7.27.1 Detailed Description

These functions implement device wide CobraNet settings. The reader is referred to "CobraNet Programmer's Reference", Cirrus Logic, <http://www.cirrus.com>, for more information.

7.27.2 Function Documentation

7.27.2.1 ASX32_API ASX_ERROR ASX_Cobranet_EnumerateModes (
ASX_HANDLE hCobranet, const int nIndex, enum
asxCOBANET_MODE * peMode, int * pnCount)

Deprecated

This function has been removed (it is stubbed out).

7.27.2.2 ASX32_API ASX_ERROR ASX_Cobranet_GetConductorPriority (
ASX_HANDLE hCobranet, unsigned int * pnPriority)

Gets a device's conductor priority.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pnPriority</i>	Returned Priority. 0 indicates that this device will never be the network conductor. 1 is the lowest priority and 0xFF is the highest priority.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.3 ASX32_API ASX_ERROR ASX_Cobranet_GetConductorStatus (
ASX_HANDLE hCobranet, unsigned int * pnState)

Gets a device's conductor status.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pnState</i>	Returned state. 1 indicates that this device is the conductor for the Cobranet network. 0 indicates that it is not the conductor.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.4 ASX32_API ASX_ERROR ASX_Cobranet_GetDescription (
ASX_HANDLE hCobranet, char * szString, const int nLength)

Get the device's description from the sysDescr SNMP field.

This is a read-only operation.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>szString</i>	Pointer to a string of length ASX_LONG_STRING.
<i>nLength</i>	Description string length.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.27.2.5 ASX32_API ASX_ERROR ASX_Cobranet_GetErrorInfo (
ASX_HANDLE hCobranet, unsigned int * pnCode, unsigned int
*** pnCount, unsigned int * pnDisplay)**

Gets a device's error information.

Returns SNMP variables errorCode, errorCount and errorDisplay.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pnCode</i>	Pointer used to return error code. See error code listing the Cobranet Programmer Manual from Cirrus Logic.
<i>pnCount</i>	Pointer used to return error count which contains the number of errors that have occurred.
<i>pnDisplay</i>	Pointer used to return error display number.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.6 ASX32_API ASX_ERROR ASX_Cobranet_GetFirmwareRevision (ASX_HANDLE *hCobranet*, char * *pszRevision*)

Gets a device's firmware revision.

This is a combination of SNMP fields `firmwareProtocolVersion`, `firmwareMajorVersion` and `firmwareMinorVersion`.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pszRevision</i>	A pointer to a char array of length <code>ASX_SHORT_STRING</code> to return the revision string.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.7 ASX32_API ASX_ERROR ASX_Cobranet_GetIfStatus (ASX_HANDLE *hCobranet*, unsigned int * *pnCurrentIf*, unsigned int * *pnPrimaryLinkStatus*, unsigned int * *pnSecondaryLinkStatus*)

Gets a device's ethernet connection status.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pnCurrentIf</i>	Returns which ethernet connection is active (1 for primary, 2 for secondary)
<i>pnPrimaryLinkStatus</i>	Returns primary link status. See asxCOBANET_IFSTATUS .
<i>pnSecondaryLinkStatus</i>	Returns secondary link status. See asxCOBANET_IFSTATUS .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.8 ASX32_API ASX_ERROR ASX_Cobranet_GetIPAddress (ASX_HANDLE *hCobranet*, unsigned int * *pdwIPAddr*)

Get the current IP address of the Cobranet device.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pdwIPAddr</i>	Gets the IP address.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.9 ASX32_API ASX_ERROR ASX_Cobranet_GetLatencyAndSampleRate (
ASX_HANDLE hCobranet, enum asxCOBANET_-
LATENCY * peLatency, enum asxSAMPLE_RATE * peRate
)

Gets a device's latency and sample reate.

Returns information from SNMP variable modeRateControl.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>peLatency</i>	Pointer used to return latency.
<i>peRate</i>	Pointer used to return sample rate.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.10 ASX32_API ASX_ERROR ASX_Cobranet_GetLocation (
ASX_HANDLE hCobranet, char * szString, const int nLength)

Get the device's location from the sysLocation SNMP field.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>szString</i>	Pointer to a string of length ASX_LONG_STRING.
<i>nLength</i>	Name string length.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.27.2.11 ASX32_API ASX_ERROR ASX_Cobranet_GetMACAddress (
ASX_HANDLE hCobranet, unsigned int * pdwMAC_MSBs,
unsigned short * pwMAC_LSBs)

Get the current cobranet MAC address.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pdwMAC_-MSBs</i>	Gets the four most significant bytes of the MAC address.
<i>pwMAC_-LSBs</i>	Gets the two least significant bytes of the MAC address.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.12 ASX32_API ASX_ERROR ASX_Cobranet_GetMode (
ASX_HANDLE hCobranet, enum asxCOBANET_MODE
*** *peMode*)**

Deprecated

This function has been removed (it is stubbed out).

7.27.2.13 ASX32_API ASX_ERROR ASX_Cobranet_GetName (
ASX_HANDLE hCobranet, char * *szString*, const int *nLength*)

Get the device's name from the sysName SNMP field.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>szString</i>	Pointer to a string of length ASX_LONG_STRING.
<i>nLength</i>	Name string length.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.27.2.14 ASX32_API ASX_ERROR ASX_Cobranet_GetPersistence (
ASX_HANDLE hCobranet, unsigned int * *pnSetting*)

Gets a device's flash persistence setting.

Returns information from SNMP variable flashPersistEnable. This should be set to maintain static IP assignments and Cobranet bundle and routing assignments through a powerdown. Conversely, clearing the persistence bit and then powering cycling the ASI2416 will set it back to factory defaults.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pnSetting</i>	0 indicates no persistence set. Non-zero indicates persistence enabled.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.15 ASX32_API ASX_ERROR ASX_Cobranet_GetSerialConfig (
ASX_HANDLE hCobranet, unsigned int * pnBaud, unsigned int
*** pnPPeriod, char pRxMAC[6], int * pnAcceptUnicast, char pTxMAC[6])**

Gets a device's serial bridge configuration.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pnBaud</i>	Returns the baud rate for the serial bridge.
<i>pnPPeriod</i>	Returns the time in 256ths of a millisecond before a character received at the SCI port is placed in a packet and transmitted.
<i>pRxMAC</i>	Gets the multicast MAC from which SCI data will be accepted.
<i>pnAcceptUnicast</i>	Gets the state of the accept unicast flag. 1 = accept, 0 = ignore
<i>pTxMAC</i>	Gets the MAC to which SCI data is sent.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.16 ASX32_API ASX_ERROR ASX_Cobranet_GetSerialEnable (
ASX_HANDLE hCobranet, int * pOnOff)

Gets a device's serial bridge status.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pOnOff</i>	Returned state. 1 is on, 0 is off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.17 ASX32_API ASX_ERROR ASX_Cobranet_GetStaticIPAddress (
ASX_HANDLE hCobranet, unsigned int * pdwIPAddr)

Get the static IP address of the Cobranet device.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pdwIPAddr</i>	Gets the static IP address. A value of 0 (0.0.0.0) indicates that no static IP address has been assigned to this device.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.18 ASX32_API ASX_ERROR ASX_Cobranet.SetConductorPriority (
ASX_HANDLE hCobranet, const unsigned int nPriority)

Sets a device's conductor priority.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>nPriority</i>	Priority to set. 0 indicates that this device will never be the network conductor. 1 is the lowest priority and 0xFF is the highest priority.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.19 ASX32_API ASX_ERROR ASX_Cobranet.SetIPAddress (
ASX_HANDLE hCobranet, const unsigned int dwIPAddr)

Set the current IP address of the Cobranet device.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>dwIPAddr</i>	The IP address to set.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.20 ASX32_API ASX_ERROR ASX_Cobranet.SetLatencyAndSampleRate (
ASX_HANDLE hCobranet, const enum
asxCOBANET_LATENCY eLatency, const enum asxSAMPLE_RATE
eRate)

Gets a device's latency and sample reate.

Returns information from SNMP variable modeRateControl.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>eLatency</i>	Latency to set.
<i>eRate</i>	Sample rate to set. Note that the ASI2416 only supports a sample rate of 48 kHz.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.21 ASX32_API ASX_ERROR ASX_Cobranet_SetLocation (
ASX_HANDLE hCobranet, const char * pszLongInputString)

Set the device's location in the sysLocation SNMP field.

Maximum name length is 60 characters.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pszLongInputString</i>	Pointer to a 0 terminated string of length ASX_LONG_STRING. Only the first 60 characters of the string should be used.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.22 ASX32_API ASX_ERROR ASX_Cobranet_SetMode (
ASX_HANDLE hCobranet, const enum
asxCOBANET_MODE eMode)**Deprecated**

This function has been removed (it is stubbed out).

7.27.2.23 ASX32_API ASX_ERROR ASX_Cobranet_SetName (
ASX_HANDLE hCobranet, const char * pszLongInputString)

Set the device's name in the sysName SNMP field.

Maximum name length is 60 characters.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>pszLongInputString</i>	Pointer to a 0 terminated string of length ASX_LONG_STRING. Only the first 60 characters of the string should be used.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.24 ASX32_API ASX_ERROR ASX_Cobranet.SetPersistence (
ASX_HANDLE hCobranet, const unsigned int nSetting)

Sets a device's flash persistence state.

Writes setting to SNMP variable flashPersistEnable. This should be set to maintain static IP assignments and Cobranet bundle and routing assignments through a power-down. Conversely, clearing the persistence bit and then powering cycling the ASI2416 will set it back to factory defaults.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>nSetting</i>	0 indicates no persistence set. Non-zero indicates persistence enabled.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.25 ASX32_API ASX_ERROR ASX_Cobranet.SetSerialConfig (
ASX_HANDLE hCobranet, const unsigned int nBaud, const
unsigned int nPPeriod, const char pRxMAC[6], const int nAcceptUnicast, const char
pTxMAC[6])

Configures a device's serial bridge.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>nBaud</i>	The baud rate for the serial bridge.
<i>nPPeriod</i>	Time in 256ths of a millisecond before a character received at the SCI port is placed in a packet and transmitted.
<i>pRxMAC</i>	Sets the multicast MAC from which SCI data will be accepted.
<i>nAcceptUnicast</i>	Set to accept properly unicast addressed data in addition to data addressed in accordance to pRxMAC. 1 = accept, 0 = ignore
<i>pTxMAC</i>	Sets the MAC address (unicast or multicast) to which SCI data is sent.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.26 ASX32_API ASX_ERROR ASX_Cobranet.SetSerialEnable (
ASX_HANDLE hCobranet, const int nOnOff)

Enable or disable a device's serial bridge.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>nOnOff</i>	1 is on, 0 is off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.27.2.27 ASX32_API ASX_ERROR ASX_Cobranet_SetStaticIPAddress (ASX_HANDLE hCobranet, const unsigned int dwIPAddr)

Set the static IP address of the Cobranet device.

Parameters

<i>hCobranet</i>	A handle to an ASX cobranet control.
<i>dwIPAddr</i>	The static IP address to set. A value of 0 resets the static IP address. The assigned static IP address will not take effect until the device is restarted.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.28 Cobranet transmitter control functions

These functions implement Cobranet transmitter configuration.

Functions

- [ASX_CobranetTx_GetStatus](#)
Gets a Cobranet transmitter's status.
- [ASX_CobranetTx_GetBundle](#)
Gets a Cobranet transmitter's bundle.
- [ASX_CobranetTx_SetBundle](#)
Sets a Cobranet transmitter's bundle.
- [ASX_CobranetTx_GetChannelCount](#)
Gets a Cobranet transmitter's channel count.
- [ASX_CobranetTx_SetChannelCount](#)
Sets a Cobranet transmitter's channel count.
- [ASX_CobranetTx_GetChannelMap](#)
Gets a Cobranet transmitter's channel map.

- [ASX_CobranetTx_SetChannelMap](#)
Sets a Cobranet transmitter's channel map.
- [ASX_CobranetTx_GetFormat](#)
Gets a Cobranet transmitter's sub format map.
- [ASX_CobranetTx_SetFormat](#)
Sets a Cobranet transmitter's channel format.
- [ASX_CobranetTx_GetUnicastMode](#)
Gets a Cobranet transmitter's unicast information.
- [ASX_CobranetTx_SetUnicastMode](#)
Sets a Cobranet transmitter's unicast information.

7.28.1 Detailed Description

These functions implement Cobranet transmitter configuration.

7.28.2 Function Documentation

7.28.2.1 ASX32_API ASX_ERROR ASX_CobranetTx_GetBundle (ASX_HANDLE *hCobranetTx*, unsigned int * *pnBundle*)

Gets a Cobranet transmitter's bundle.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>pnBundle</i>	The transmitter's assigned bundle. A value of 0 indicates that the transmitter is disabled.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.28.2.2 ASX32_API ASX_ERROR ASX_CobranetTx_GetChannelCount (ASX_HANDLE *hCobranetTx*, unsigned int * *pnCount*)

Gets a Cobranet transmitter's channel count.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>pnCount</i>	The number of audio channels to transmit in a bundle. This is SNMP variable txSubCount.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.28.2.3 ASX32_API ASX_ERROR ASX_CobranetTx_GetChannelMap (ASX_HANDLE hCobranetTx, unsigned int nMap[8])

Gets a Cobranet transmitter's channel map.

This controls which audio routing channels are transmitted.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>nMap</i>	The channel map returned in an array of size 8.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.28.2.4 ASX32_API ASX_ERROR ASX_CobranetTx_GetFormat (ASX_HANDLE hCobranetTx, enum asxAUDIO_FORMAT * peFormat)

Gets a Cobranet transmitter's sub format map.

This contains transmit format information.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>peFormat</i>	The format returned, will be asxAUDIO_FORMAT_PCM16, asxAUDIO_FORMAT_PCM20, asxAUDIO_FORMAT_PCM24 or asxAUDIO_FORMAT_NONE.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.28.2.5 ASX32_API ASX_ERROR ASX_CobranetTx_GetStatus (
ASX_HANDLE hCobranetTx, unsigned int * pnDropouts,
unsigned int * pnPosition, unsigned int * pnReceivers)

Gets a Cobranet transmitter's status.

This function reads SNMP variables txDropouts, txPosition, txReceivers.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>pnDropouts</i>	Number of times transmission has been interrupted.
<i>pnPosition</i>	Transmission permission position.
<i>pnReceivers</i>	Number of receivers requesting the bundle transmitted by this transmitter.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.28.2.6 ASX32_API ASX_ERROR ASX_CobranetTx_GetUnicastMode (
ASX_HANDLE hCobranetTx, unsigned int * pnUnicastMode,
unsigned int * pnMaxUnicast)

Gets a Cobranet transmitter's unicast information.

This function reads SNMP variable txUnicastMode.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>pnUnicastMode</i>	Specifies the number of unicast destinations served before switching to multicast.
<i>pnMaxUnicast</i>	Specifies the maximum number of unicast destinations.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.28.2.7 ASX32_API ASX_ERROR ASX_CobranetTx_SetBundle (
ASX_HANDLE hCobranetTx, const unsigned int nBundle)

Sets a Cobranet transmitter's bundle.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>nBundle</i>	The bundle to assign to the transmitter. A value of 0 indicates that the transmitter is disabled.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.28.2.8 ASX32_API ASX_ERROR ASX_CobranetTx_SetChannelCount (
ASX_HANDLE hCobranetTx, const unsigned int nCount)

Sets a Cobranet transmitter's channel count.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>nCount</i>	The number of audio channels to transmit in a bundle. This is SNMP variable txSubCount.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.28.2.9 ASX32_API ASX_ERROR ASX_CobranetTx_SetChannelMap (
ASX_HANDLE hCobranetTx, const unsigned int nMap[8])

Sets a Cobranet transmitter's channel map.

This controls which audio routing channels are transmitted.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>nMap</i>	The channel map is passed as in an array of size 8.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.28.2.10 ASX32_API ASX_ERROR ASX_CobranetTx_SetFormat (
ASX_HANDLE hCobranetTx, const enum
asxAUDIO_FORMAT eFormat)

Sets a Cobranet transmitter's channel format.

This controls the resolution, sample rate and latency of each transmitted channel. In practice all channels are set to the same value.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.
<i>eFormat</i>	The channel format specification, can be <code>asxAUDIO_FORMAT_PCM16</code> , <code>asxAUDIO_FORMAT_PCM20</code> or <code>asxAUDIO_FORMAT_PCM24</code> . Specify <code>asxAUDIO_FORMAT_NONE</code> to disable the transmitter (same effect as <code>bundle==0</code>).

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.28.2.11 ASX32_API ASX_ERROR ASX_CobranetTx_SetUnicastMode (ASX_HANDLE hCobranetTx, const unsigned int nUnicastMode, const unsigned int nMaxUnicast)

Sets a Cobranet transmitter's unicast information.

This function writes SNMP variable `txUnicastMode`.

Parameters

<i>hCobranetTx</i>	A handle to an ASX cobranet transmitter control.		
<i>nUnicastMode</i>	Specifies the number of unicast destinations served before switching to multicast. Value of <code>0x7FFFFFFF</code> disables multicast addressing.		
<i>nMaxUnicast</i>	Specifies the maximum number of unicast destinations.		
	nUnicastMode value	nMaxUnicast value	Effect
	0	any	Always Multicast
	0x7FFFFFFF	N (1 to 4)	Up to N receivers unicast. Additional receivers fail.
	N	N	Up to N receivers unicast switch to multicast for more.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.29 Cobranet receiver control functions

These functions implement Cobranet receiver configuration.

Functions

- [ASX_CobranetRx_GetStatus](#)
Gets a Cobranet receiver's status.
- [ASX_CobranetRx_GetBundle](#)
Gets a Cobranet receiver's bundle.
- [ASX_CobranetRx_SetBundle](#)
Sets a Cobranet receiver's bundle.
- [ASX_CobranetRx_GetSourceMAC](#)
Gets a Cobranet receiver's source MAC address for private bundles.
- [ASX_CobranetRx_SetSourceMAC](#)
Sets a Cobranet receiver's source MAC address for private bundles.
- [ASX_CobranetRx_GetChannelMap](#)
Gets a Cobranet receiver's channel mapping.
- [ASX_CobranetRx_SetChannelMap](#)
Sets a Cobranet receiver's channel mapping.
- [ASX_CobranetRx_GetMinimumDelay](#)
Gets a Cobranet receiver's minimum delay.
- [ASX_CobranetRx_SetMinimumDelay](#)
Sets a Cobranet receiver's channel mapping.

7.29.1 Detailed Description

These functions implement Cobranet receiver configuration.

7.29.2 Function Documentation

7.29.2.1 ASX32_API ASX_ERROR ASX_CobranetRx_GetBundle (ASX_HANDLE hCobranetRx, unsigned int * pnBundle)

Gets a Cobranet receiver's bundle.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>pnBundle</i>	Bundle number being received. 0 indicates no bundle is being received.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.29.2.2 ASX32_API ASX_ERROR ASX_CobranetRx.GetChannelMap (ASX_HANDLE *hCobranetRx*, unsigned int *nMap[8]*)

Gets a Cobranet receiver's channel mapping.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>nMap</i>	Audio channel mapping.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Examples:

[cobranet/main.c](#).

7.29.2.3 ASX32_API ASX_ERROR ASX_CobranetRx.GetMinimumDelay (ASX_HANDLE *hCobranetRx*, unsigned int * *pnMinDelay*)

Gets a Cobranet receiver's minimum delay.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>pnMinDelay</i>	Minimum received audio delay.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.29.2.4 ASX32_API ASX_ERROR ASX_CobranetRx.GetSourceMAC (ASX_HANDLE *hCobranetRx*, unsigned int * *pdwMAC_MSBs*, unsigned short * *pwMAC_LSBs*)

Gets a Cobranet receiver's source MAC address for private bundles.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>pdwMAC_MSBs</i>	Gets the four most significant bytes of the source MAC address.
<i>pdwMAC_LSBs</i>	Gets the two least significant bytes of the source MAC address.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.29.2.5 ASX32 API ASX_ERROR ASX_CobranetRx_GetStatus (
ASX_HANDLE hCobranetRx, unsigned int * pnStatus, unsigned
int * pnDropouts, unsigned int * pnDelay, unsigned int nFormat[8])

Gets a Cobranet receiver's status.

This function reads SNMP variables rxStatus, rxDropouts, rxDelay and rxSubFormat.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>pnStatus</i>	1 indicates a bundle is being received. 0 indicates no bundle is being received.
<i>pnDropouts</i>	Counts the number of times bundle reception has been interrupted.
<i>pnDelay</i>	A non-zero value indicates the number of additional delays imposed on the received audio due to network forwarding delays. Delay is expressed in units of the system latency. ie 1-1/3ms for 5-1/3ms latency.

<i>nFormat</i>	The format information for the received audio channels. Subformat codes are shown in the following table. The least significant bit of the subformat entries is set when the received format is supported for reception by the CobraNet interface. A test of this least significant bit can be used to determine correct reception on a per audio channel basis.		
txSubFormat value	Resolution	Sample Rate	Latency
0	invalid	invalid	invalid
0x044000	16-bit	48 kHz	5-1/3 ms
0x054000	20-bit	48 kHz	5-1/3 ms
0x064000	24-bit	48 kHz	5-1/3 ms
0x148000	16-bit	96 kHz	5-1/3 ms
0x158000	20-bit	96 kHz	5-1/3 ms
0x168000	24-bit	96 kHz	5-1/3 ms
0x042000	16-bit	48 kHz	2-2/3 ms
0x052000	20-bit	48 kHz	2-2/3 ms
0x062000	24-bit	48 kHz	2-2/3 ms
0x144000	16-bit	96 kHz	2-2/3 ms
0x154000	20-bit	96 kHz	2-2/3 ms
0x164000	24-bit	96 kHz	2-2/3 ms
0x041000	16-bit	48 kHz	1-1/3 ms
0x051000	20-bit	48 kHz	1-1/3 ms
0x061000	24-bit	48 kHz	1-1/3 ms
0x142000	16-bit	96 kHz	1-1/3 ms
0x152000	20-bit	96 kHz	1-1/3 ms
0x162000	24-bit	96 kHz	1-1/3 ms

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.29.2.6 ASX32_API ASX_ERROR ASX_CobranetRx.SetBundle (ASX_HANDLE hCobranetRx, const unsigned int nBundle)

Sets a Cobranet receiver's bundle.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>nBundle</i>	Bundle number to receive. 0 indicates no bundle is being received.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.29.2.7 ASX32 API ASX_ERROR ASX_CobranetRx_SetChannelMap (
ASX_HANDLE *hCobranetRx*, const unsigned int *nMap*[8])

Sets a Cobranet receiver's channel mapping.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>nMap</i>	Audio channel mapping.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.29.2.8 ASX32 API ASX_ERROR ASX_CobranetRx_SetMinimumDelay (
ASX_HANDLE *hCobranetRx*, const unsigned int *nMinDelay*)

Sets a Cobranet receiver's channel mapping.

Selects a minimum additional delay imposed on the received audio. Delay is expressed in units of isochronous cycles (1-1/3ms for standard 5-1/3ms latency mode, 2/3ms for 2-2/3ms latency mode and 1/3ms for 1-1/3ms latency mode). This variable is designed to allow configuration of a deterministic common delay for all Cobranet interfaces in larger network installations. rxDelay will never be reduced below this setting. This variable is not designed for actively delaying audio for architectural applications.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>nMinDelay</i>	Minium audio delay.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.29.2.9 ASX32 API ASX_ERROR ASX_CobranetRx_SetSourceMAC (
ASX_HANDLE *hCobranetRx*, const unsigned int *dwMAC_MSBs*,
const unsigned short *wMAC_LSBs*)

Sets a Cobranet receiver's source MAC address for private bundles.

Parameters

<i>hCobranetRx</i>	A handle to an ASX cobranet receiver control.
<i>dwMAC_MSBs</i>	Sets the four most significant bytes of the source MAC address.
<i>wMAC_LSBs</i>	Sets the two least significant bytes of the source MAC address.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.30 Tone detector control functions

The tone detector monitors its inputs for the presence of any of a number of tones.

Functions

- [ASX_ToneDetector_SetEnable](#)
Turns the entire tone detector on and off.
- [ASX_ToneDetector_GetEnable](#)
Returns whether the entire tone detector is on or off.
- [ASX_ToneDetector_SetEventEnable](#)
Turns the event reporting function of the tone detector on and off.
- [ASX_ToneDetector_GetEventEnable](#)
Returns whether the event reporting function of the tone detector is on or off.
- [ASX_ToneDetector_SetThreshold](#)
Sets the tone detector threshold (units of dB)
- [ASX_ToneDetector_GetThreshold](#)
Gets the tone detector threshold (units of dB) with respect to full scale eg.
- [ASX_ToneDetector_GetState](#)
Gets the tone detector state.
- [ASX_ToneDetector_GetFrequency](#)
Gets the centre frequency of each tone detector by index.

7.30.1 Detailed Description

The tone detector monitors its inputs for the presence of any of a number of tones. Currently 25Hz and 35Hz tones can be detected independently on left and right channels. Tones that exceed the threshold set by `HPI_ToneDetector_SetThreshold()` are detected. The result of the detection is reflected in the controls state, and optionally by sending an async event with the new state. Tones must have a minimum duration of 200ms before they are detected.

7.30.2 Function Documentation

7.30.2.1 ASX32_API ASX_ERROR ASX_ToneDetector_GetEnable (
ASX_HANDLE *hToneDetector*, unsigned int * *nEnable*)

Returns whether the entire tone detector is on or off.

Parameters

<i>hToneDetector</i>	A handle to an ASX tone detector control.
<i>nEnable</i>	A return value of 1 implies on, while 0 implies off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.30.2.2 ASX32_API ASX_ERROR ASX_ToneDetector_GetEventEnable (
ASX_HANDLE *hToneDetector*, unsigned int * *nEnable*)

Returns whether the event reporting function of the tone detector is on or off.

Parameters

<i>hToneDetector</i>	A handle to an ASX tone detector control.
<i>nEnable</i>	A return value of 1 implies on, while 0 implies off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.30.2.3 ASX32_API ASX_ERROR ASX_ToneDetector_GetFrequency (
ASX_HANDLE *hToneDetector*, unsigned int *nIndex*, unsigned int
*** *nState*)**

Gets the centre frequency of each tone detector by index.

This can be used to determine the meanings of the state bits.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.30.2.4 ASX32_API ASX_ERROR ASX_ToneDetector_GetState (
ASX_HANDLE *hToneDetector*, unsigned int * *nState*)

Gets the tone detector state.

The state is a bitfield. Pairs of bits represent left and right channels of detectors.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.30.2.5 ASX32_API ASX_ERROR ASX_ToneDetector_GetThreshold (
ASX_HANDLE *hToneDetector*, float * *fThreshold*)

Gets the tone detector threshold (units of dB) with respect to full scale eg. -20dBFS. Tones with level above -20dBFS threshold are detected Tones must have a minimum duration of 200ms before they are detected.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.30.2.6 ASX32_API ASX_ERROR ASX_ToneDetector_SetEnable (
ASX_HANDLE *hToneDetector*, const unsigned int *nEnable*)

Turns the entire tone detector on and off.

Parameters

<i>hToneDetector</i>	A handle to an ASX tone detector control.
<i>nEnable</i>	A value of 1 enables the tone detector and a value of 0 disables it.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.30.2.7 ASX32_API ASX_ERROR ASX_ToneDetector_SetEventEnable (
ASX_HANDLE *hToneDetector*, const unsigned int *nEnable*)

Turns the event reporting function of the tone detector on and off.

Parameters

<i>hToneDetector</i>	A handle to an ASX tone detector control.
<i>nEnable</i>	A value of 1 enables the tone detector event reporting and a value of 0 disables it.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.30.2.8 ASX32 API `ASX_ERROR ASX_ToneDetector_SetThreshold ( ASX_HANDLE hToneDetector, const float fThreshold )`

Sets the tone detector threshold (units of dB)

Returns

Returns 0 if there is no error, otherwise one of `asxERROR` is returned.

7.31 Silence detector control functions

These functions control a silence detector.

Functions

- [`ASX\_SilenceDetector\_SetEnable`](#)
Turns the entire silence detector on and off.
- [`ASX\_SilenceDetector\_GetEnable`](#)
Returns whether the entire silence detector is on or off.
- [`ASX\_SilenceDetector\_SetEventEnable`](#)
Turns the event reporting function of the silence detector on and off.
- [`ASX\_SilenceDetector\_GetEventEnable`](#)
Returns whether the event reporting function of the silence detector is on or off.
- [`ASX\_SilenceDetector\_SetDelay`](#)
Set the silence detector delay.
- [`ASX\_SilenceDetector\_GetDelay`](#)
Get the silence detector delay.
- [`ASX\_SilenceDetector\_SetThreshold`](#)
Sets the silence detector threshold (units of dB)
- [`ASX\_SilenceDetector\_GetThreshold`](#)
Gets the silence detector threshold (units of dB)
- [`ASX\_SilenceDetector\_GetState`](#)
Gets the silence detector state.

7.31.1 Detailed Description

These functions control a silence detector.

7.31.2 Function Documentation

7.31.2.1 ASX32_API ASX_ERROR ASX_SilenceDetector_GetDelay (ASX_HANDLE *hSilenceDetector*, unsigned int * *Delay*)

Get the silence detector delay.

Parameters

<i>hSilenceDetector</i>	A handle to an ASX silence detector control.
<i>Delay</i>	Delay in milliseconds after signal falls below threshold before silence is indicated.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.31.2.2 ASX32_API ASX_ERROR ASX_SilenceDetector_GetEnable (ASX_HANDLE *hSilenceDetector*, unsigned int * *nEnable*)

Returns whether the entire silence detector is on or off.

Parameters

<i>hSilenceDetector</i>	A handle to an ASX silence detector control.
<i>nEnable</i>	A return value of 1 implies on, while 0 implies off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.31.2.3 ASX32_API ASX_ERROR ASX_SilenceDetector_GetEventEnable (ASX_HANDLE *hSilenceDetector*, unsigned int * *nEnable*)

Returns whether the event reporting function of the silence detector is on or off.

Parameters

<i>hSilenceDetector</i>	A handle to an ASX silence detector control.
<i>nEnable</i>	A return value of 1 implies on, while 0 implies off.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.31.2.4 ASX32_API ASX_ERROR ASX_SilenceDetector_GetState (
ASX_HANDLE *hSilenceDetector*, unsigned int * *nState*)

Gets the silence detector state.

The state is a bitfield. Pairs of bits represent left and right channels of detectors.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.31.2.5 ASX32_API ASX_ERROR ASX_SilenceDetector_GetThreshold (
ASX_HANDLE *hSilenceDetector*, float * *fThreshold*)

Gets the silence detector threshold (units of dB)

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.31.2.6 ASX32_API ASX_ERROR ASX_SilenceDetector_SetDelay (
ASX_HANDLE *hSilenceDetector*, const unsigned int *Delay*)

Set the silence detector delay.

Parameters

<i>hSilenceDetector</i>	A handle to an ASX silence detector control.
<i>Delay</i>	Delay in milliseconds after signal falls below threshold before silence is indicated.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.31.2.7 ASX32_API ASX_ERROR ASX_SilenceDetector_SetEnable (
ASX_HANDLE *hSilenceDetector*, const unsigned int *nEnable*)

Turns the entire silence detector on and off.

Parameters

<i>hSilenceDetector</i>	A handle to an ASX tone detector control.
<i>nEnable</i>	A value of 1 enables the silence detector and a value of 0 disables it.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.31.2.8 ASX32_API ASX_ERROR ASX_SilenceDetector_SetEventEnable (
ASX_HANDLE hSilenceDetector, const unsigned int nEnable)

Turns the event reporting function of the silence detector on and off.

Parameters

<i>hSilenceDetector</i>	A handle to an ASX silence detector control.
<i>nEnable</i>	A value of 1 enables the silence detector event reporting and a value of 0 disables it.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.31.2.9 ASX32_API ASX_ERROR ASX_SilenceDetector_SetThreshold (
ASX_HANDLE hSilenceDetector, const float fThreshold)

Sets the silence detector threshold (units of dB)

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32 Block functions.

These functions provide an abstract mechanism to manipulate a well defined set of property primitives that can be combined into a block.

Functions

- [ASX_Block_GetInfo](#)
Gets the name of the block and the number of parameters it has.
- [ASX_Block_Parameter_GetName](#)
Gets the name of parameter number uParameterIndex.
- [ASX_Block_Parameter_GetUnits](#)
Gets the units of parameter number uParameterIndex.
- [ASX_Block_Parameter_GetType](#)

Gets the type of parameter number `uParameterIndex`.

- [ASX_Block_Parameter_GetFlags](#)

Gets the flags for parameter number `uParameterIndex`.

- [ASX_Block_Parameter_GetElementCount](#)

Gets the number of elements.

- [ASX_Block_Parameter_GetRange](#)

Gets the parameter range.

- [ASX_Block_Parameter_GetEnumName](#)

Gets the enumerated names for a parameter.

- [ASX_Block_Parameter_Set](#)

Sets a parameter's value field.

- [ASX_Block_Parameter_Get](#)

Gets a parameter's value field.

7.32.1 Detailed Description

These functions provide an abstract mechanism to manipulate a well defined set of property primitives that can be combined into a block.

7.32.2 Function Documentation

7.32.2.1 ASX32 API ASX_ERROR ASX_Block_GetInfo (
ASX_HANDLE *hBlock*, char * *szBlockName*, const unsigned int
***uStringLength*, unsigned int * *uParameterCount*)**

Gets the name of the block and the number of parameters it has.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>szBlock-Name</i>	String used to return the name of the block. It should be of length ASX_SHORT_STRING.
<i>uS-tringLength</i>	The length on the string pointed to be <i>szName</i> .
<i>uParameter-Count</i>	The number of parameters it has.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.2 ASX32_API ASX_ERROR ASX_Block_Parameter_Get (
ASX_HANDLE *hBlock*, const unsigned int *uParameterIndex*,
struct asxParameterValue * *data*)

Gets a parameter's value field.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>data</i>	The data to get from the parameter. See asxParameterValue .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.3 ASX32_API ASX_ERROR ASX_Block_Parameter_GetElementCount (
ASX_HANDLE *hBlock*, const unsigned int *uParameterIndex*,
unsigned int * *uCount*)

Gets the number of elements.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>uCount</i>	Returned element count.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.4 ASX32_API ASX_ERROR ASX_Block_Parameter_GetEnumName (
ASX_HANDLE *hBlock*, const unsigned int *uParameterIndex*,
const unsigned int *uEnumItem*, char * *szEnumName*, const unsigned int
***uStringLength*)**

Gets the enumerated names for a parameter.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>uEnumItem</i>	The enumerator index.
<i>szEnumName</i>	String used to return the enumerator name. It should be of length ASX_SHORT_STRING.

<i>uStringLength</i>	The length on the string pointed to be szName.
----------------------	--

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.5 ASX32_API ASX_ERROR ASX_Block_Parameter_GetFlags (
ASX_HANDLE hBlock, const unsigned int uParameterIndex,
enum asxUCONTROL_PFLAGS * eFlags)

Gets the flags for parameter number uParameterIndex.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>eFlags</i>	Returned parameter flags. See asxUCONTROL_PFLAGS .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.6 ASX32_API ASX_ERROR ASX_Block_Parameter_GetName (
ASX_HANDLE hBlock, const unsigned int uParameterIndex,
char * szParameterName, const unsigned int uStringLength)

Gets the name of parameter number uParameterIndex.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>szParameterName</i>	String used to return the name of the parameter. It should be of length ASX_SHORT_STRING.
<i>uStringLength</i>	The length on the string pointed to be szName.

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.7 ASX32_API ASX_ERROR ASX_Block_Parameter_GetRange (
ASX_HANDLE *hBlock*, const unsigned int *uParameterIndex*,
struct asxParameterRangeInfo * *info*)

Gets the parameter range.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>info</i>	Returned range structure. See asxParameterRangeInfo .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.8 ASX32_API ASX_ERROR ASX_Block_Parameter_GetType (
ASX_HANDLE *hBlock*, const unsigned int *uParameterIndex*,
enum asxUCONTROL_PTYPE * *eType*)

Gets the type of parameter number *uParameterIndex*.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>eType</i>	Returned parameter type. See asxUCONTROL_PTYPE .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.9 ASX32_API ASX_ERROR ASX_Block_Parameter_GetUnits (
ASX_HANDLE *hBlock*, const unsigned int *uParameterIndex*,
char * *szParameterUnits*, const unsigned int *uStringLength*)

Gets the units of parameter number *uParameterIndex*.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>szParameterUnits</i>	String used to return the parameter units. It should be of length ASX_SHORT_STRING.
<i>uStringLength</i>	The length on the string pointed to be <i>szName</i> .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

7.32.2.10 ASX32_API ASX_ERROR ASX_Block_Parameter_Set (
 ASX_HANDLE *hBlock*, const unsigned int *uParameterIndex*,
 struct asxParameterValue * *data*)

Sets a parameter's value field.

Parameters

<i>hBlock</i>	A handle to an ASX block.
<i>uParameterIndex</i>	The index of the parameter of interest.
<i>data</i>	The data to set on the parameter. See asxParameterValue .

Returns

Returns 0 if there is no error, otherwise one of [asxERROR](#) is returned.

Chapter 8

Data Structure Documentation

8.1 asxCobranetIpAutoassignParameters Struct Reference

```
#include <asx.h>
```

Data Fields

- char [addr_start](#) [20]
- char [addr_end](#) [20]
- int [autoassign](#)

8.1.1 Field Documentation

8.1.1.1 char [asxCobranetIpAutoassignParameters::addr_end](#)[20]

8.1.1.2 char [asxCobranetIpAutoassignParameters::addr_start](#)[20]

8.1.1.3 int [asxCobranetIpAutoassignParameters::autoassign](#)

The documentation for this struct was generated from the following file:

- [asx.h](#)

8.2 asxParameterRangeInfo Struct Reference

```
#include <asx.h>
```

Data Fields

- enum [asxUCONTROL_RTYPE](#) type

```
• union {  
    struct {  
        int min  
        int max  
        int step  
    } integer  
    struct {  
        unsigned int count  
        int * value  
    } enumerated_integer  
    struct {  
        unsigned int count  
        float * value  
    } enumerated_float  
    struct {  
        unsigned int count  
        struct asxParameterRangeInfo_NamedEnumerated * enums  
    } enumerated  
    struct {  
        float fmin  
        float fmax  
        float fstep  
    } floating  
    struct {  
        unsigned int max_len  
    } string  
} u
```

8.2.1 Field Documentation

- 8.2.1.1 unsigned int asxParameterRangeInfo::count
- 8.2.1.2 struct { ... } asxParameterRangeInfo::enumerated
- 8.2.1.3 struct { ... } asxParameterRangeInfo::enumerated_float
- 8.2.1.4 struct { ... } asxParameterRangeInfo::enumerated_integer
- 8.2.1.5 struct asxParameterRangeInfo_NamedEnumerated*
asxParameterRangeInfo::enums
- 8.2.1.6 struct { ... } asxParameterRangeInfo::floating
- 8.2.1.7 float asxParameterRangeInfo::fmax
- 8.2.1.8 float asxParameterRangeInfo::fmin
- 8.2.1.9 float asxParameterRangeInfo::fstep
- 8.2.1.10 struct { ... } asxParameterRangeInfo::integer
- 8.2.1.11 int asxParameterRangeInfo::max
- 8.2.1.12 unsigned int asxParameterRangeInfo::max_len
- 8.2.1.13 int asxParameterRangeInfo::min
- 8.2.1.14 int asxParameterRangeInfo::step
- 8.2.1.15 struct { ... } asxParameterRangeInfo::string
- 8.2.1.16 enum asxUCONTROL_RTYPE asxParameterRangeInfo::type
- 8.2.1.17 union { ... } asxParameterRangeInfo::u
- 8.2.1.18 float* asxParameterRangeInfo::value
- 8.2.1.19 int* asxParameterRangeInfo::value

The documentation for this struct was generated from the following file:

- [asx.h](#)

8.3 asxParameterRangeInfo_NamedEnumerated Struct Reference

```
#include <asx.h>
```

Data Fields

- int [value](#)
- char [name](#) [ASX_SHORT_STRING]

8.3.1 Field Documentation

8.3.1.1 char [asxParameterRangeInfo_NamedEnumerated::name](#)[ASX_SHORT_STRING]

8.3.1.2 int [asxParameterRangeInfo_NamedEnumerated::value](#)

The documentation for this struct was generated from the following file:

- [asx.h](#)

8.4 asxParameterValue Struct Reference

```
#include <asx.h>
```

Data Fields

- enum [asxUCONTROL_PTYPE](#) [eType](#)
- unsigned int [uItems](#)
- size_t [size](#)
- void * [value](#)

8.4.1 Field Documentation

8.4.1.1 enum [asxUCONTROL_PTYPE](#) [asxParameterValue::eType](#)

8.4.1.2 size_t [asxParameterValue::size](#)

8.4.1.3 unsigned int [asxParameterValue::uItems](#)

8.4.1.4 void* [asxParameterValue::value](#)

The documentation for this struct was generated from the following file:

- [asx.h](#)

Chapter 9

File Documentation

9.1 asx.h File Reference

Data Structures

- struct [asxParameterRangeInfo_NamedEnumerated](#)
- struct [asxParameterRangeInfo](#)
- struct [asxParameterValue](#)
- struct [asxCobranetIpAutoassignParameters](#)

Defines

- #define [ARRAY_SIZE\(X\)](#) (sizeof(X)/sizeof(X[0]))
- #define [ASX32_API](#)
- #define [ASX_SYSTEM_TYPE_HPI](#) 0
Use this to select ASI's HPI interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_WAVE](#) 1
Use this to select Microsoft's WAVE interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_ALSA](#) 2
Use this to select the Linux ALSA interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_DIRECTX](#) 3
Use this to select Microsoft's DirectX interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_PORTAUDIO](#) 4
Use this to select the PortAudio interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_PCXTOOLS](#) 5
Use this to select Digigram's PCX interface when calling [ASX_System_Create\(\)](#).

- #define [ASX_SYSTEM_TYPE_SNMP](#) 6
Use this to select Cobranet SNMP interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_HPIUDP](#) 7
Use this to select ASI's HPI interface over UDP when calling [ASX_System_Create\(\)](#). Also supports HPI.
- #define [ASX_SYSTEM_TYPE_DUMMY](#) 8
Dummy backend.
- #define [ASX_SYSTEM_TYPE_ANY](#) 9
Wild card - any subsystem (reserved).
- #define [ASX_SYSTEM_TYPE_ASIO](#) 10
Use this to select Stienberg's ASIO interface when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_AVB_1722_1](#) 11
Use this to select the IEEE 1722.1 system for controlling AVB devices when calling [ASX_System_Create\(\)](#).
- #define [ASX_SYSTEM_TYPE_COUNT](#) 12
Indicates the number of subsystems defined.
- #define [ASX_SHORT_STRING](#) 32
Short string size for adapter, node, control, enum translations.
- #define [ASX_LONG_STRING](#) 128
Long string size for error strings, filenames and PADs strings.
- #define [ASX_LOGLONG_STRING](#) 256
LongLong string size for PADs comment string.
- #define [_RPT0](#)(l, s) printf(s)
A debug helper function, 0 arguments.
- #define [_RPT1](#)(l, s, d1) printf(s,d1)
A debug helper function, 1 argument.

Typedefs

- typedef void * [ASX_HANDLE](#)
Generic handle used to represent all ASX objects.
- typedef enum [asxERROR](#) [ASX_ERROR](#)

Error type used to return error codes from all functions.

- typedef enum [asxNODE ASX_NODE](#)
Node type enum.
- typedef int [ASX_TIME](#)
Timescale.
- typedef void [ASX_ERROR_CALLBACK](#) ([ASX_HANDLE](#) hASX_Object, const char *pszCallingFunction, void *pUser1, void *pUser2)
An error handling callback function.
- typedef void [ASX_PLAYER_CALLBACK](#) ([ASX_HANDLE](#) hASX_Player_Object, const enum [asxPLAYER_FLAGS](#) flags, void *pUser1)
A playback callback function.

Enumerations

- enum [asxERROR](#) {
[asxERROR_NO_ERROR](#) = 0, [asxERROR_ASXObject](#) = 256, [asxERROR_INDEX_OUT_OF_RANGE](#) = 257, [asxERROR_UNIMPLEMENTED](#) = 258,
[asxERROR_COMMUNICATING_WITH_DEVICE](#) = 259, [asxERROR_STARTING_DEVICE](#) = 260, [asxERROR_NOT_OPEN](#) = 261, [asxERROR_ALREADY_OPEN](#) = 262,
[asxERROR_INVALID_FORMAT](#) = 263, [asxERROR_INTERNAL_BUFFERING_ERROR](#) = 264, [asxERROR_AES18](#) = 265, [asxERROR_INVALID_OPERATION](#) = 266,
[asxERROR_ENUMERATE_INDEX_OUT_OF_RANGE](#) = 267, [asxERROR_BUFFER_TOO_SMALL](#) = 268, [asxERROR_OUTOFMEMORY](#) = 269, [asxERROR_DEPRECATED](#) = 270,
[asxERROR_TOO_MANY_CLIENTS](#) = 271, [asxERROR_COBRANET_NODE_NOT_FOUND](#) = 272, [asxERROR_COBRANET_NODE_FOUND](#) = 273, [asxERROR_NO_IP_ADDRESSES_AVAILABLE](#) = 274,
[asxERROR_IP_ASSIGNED](#) = 275, [asxERROR_IP_CHANGED](#) = 276, [asxERROR_IP_AUTOASSIGN_DISABLED](#) = 277, [asxERROR_PCAP_ERROR](#) = 278,
[asxERROR_DISCO_DLL_NOT_FOUND](#) = 279, [asxERROR_HOST_NOT_FOUND](#) = 280, [asxERROR_COBRANET_NODE_UNREACHABLE](#) = 281, [asxERROR_DUPLICATE_ADAPTER_INDEX](#) = 282,
[asxERROR_INVALID_CONTROL](#) = 304, [asxERROR_INVALID_CONTROL_VALUE](#) = 305, [asxERROR_INVALID_CONTROL_NOT_FOUND](#) = 306, [asxERROR_INVALID_NUMBER_OF_CHANNELS](#) = 307,
[asxERROR_INVALID_CONTROL_ATTRIBUTE](#) = 308, [asxERROR_UNSUPPORTED_CONTROL_ATTRIBUTE](#) = 309, [asxERROR_INVALID_CONTROL_OPERATION](#) = 310, [asxERROR_CONTROL_NOT_READY](#) = 311,

```

asxERROR_FILE_OPEN_FAILED = 336, asxERROR_PLAYER_INTERNAL_-
STATE_FAILURE = 384, asxERROR_PLAYER_TIME_OUT = 385, asxERROR_-
PLAYER_OUT_OF_SEQUENCE_CALL = 386,
asxERROR_PLAYER_TWAV = 387, asxERROR_PLAYER_NOFILE = 388, asxERROR_-
PLAYER_INVALIDFILEFORMAT = 389, asxERROR_PLAYER_UNSUPPORTEDFORMAT
= 390,
asxERROR_PLAYER_FILEREADERROR = 391, asxERROR_PLAYER_FILEOPENERROR
= 392, asxERROR_RECORDER_INTERNAL_STATE_FAILURE = 448, asxERROR_-
RECORDER_TIME_OUT = 449,
asxERROR_RECORDER_OUT_OF_SEQUENCE_CALL = 450, asxERROR_-
RECORDER_TWAV = 451, asxERROR_RECORDER_FILECREATEERROR
= 452, asxERROR_RECORDER_FILEWRITEERROR = 453,
asxERROR_RECORDER_FORMATMISMATCH = 454, asxERROR_RECORDER_-
INVALIDFILENAME = 455, asxERROR_MIXER_SAVECONTROLSTATE =
460, asxERROR_UNKNOWN = 4095 }

```

ASX error codes. These error codes are returned by most ASX functions.

- enum `asxNODE` {


```

asxNODE_NONE = 0, asxNODE_INVALID = 400, asxNODE_ADAPTER =
401, asxNODE_PLAYER = 402,
asxNODE_LINE_IN = 403, asxNODE_AESEBU_IN = 404, asxNODE_TUNER_-
IN = 405, asxNODE_RADIO_FREQ_IN = 406,
asxNODE_CLOCK_SOURCE_IN = 407, asxNODE_BITSTREAM_IN = 408,
asxNODE_MICROPHONE_IN = 409, asxNODE_COBRANET_IN = 410,
asxNODE_COBRANET_RECEIVER = 411, asxNODE_ANALOG_IN = 412,
asxNODE_SDI_IN = 413, asxNODE_RTP_DESTINATION_IN = 414,
asxNODE_INTERNAL_IN = 416, asxNODE_AVB_IN_STREAM = 417, asxNODE_-
BLULINK_IN = 418, asxNODE_AVB_IN_AUDIO = 419,
asxNODE_LAST_SOURCE_NODE = 420, asxNODE_FIRST_DEST_NODE
= 450, asxNODE_RECORDER = 450, asxNODE_LINE_OUT = 451,
asxNODE_AESEBU_OUT = 452, asxNODE_RADIO_FREQ_OUT = 453, asxNODE_-
SPEAKER_OUT = 454, asxNODE_COBRANET_OUT = 455,
asxNODE_COBRANET_TRANSMITTER = 456, asxNODE_ANALOG_OUT
= 457, asxNODE_SDI_OUT = 458, asxNODE_RTP_SOURCE_OUT = 459,
asxNODE_AVB_OUT_STREAM = 460, asxNODE_INTERNAL_OUT = 461,
asxNODE_BLULINK_OUT = 462, asxNODE_AVB_OUT_AUDIO = 463,
asxNODE_LAST_DEST_NODE = 464 }

```

*Node type identifiers. The nodes identify how controls are connected and located.
This enum is used to identify node types.*

- enum `asxCONTROL` {


```

asxCONTROL_INVALID = 500, asxCONTROL_CONNECTION = 501, asxCONTROL_-
VOLUME = 502, asxCONTROL_METER = 503,
asxCONTROL_MUTE = 504, asxCONTROL_MULTIPLEXER = 505, asxCONTROL_-
AESEBU_TRANSMITTER = 506, asxCONTROL_AESEBU_RECEIVER = 507,

```

```

asxCONTROL_LEVEL = 508, asxCONTROL_TUNER = 509, asxCONTROL_-
RDS = 510, asxCONTROL_VOX = 511,
asxCONTROL_AES18_TRANSMITTER = 512, asxCONTROL_AES18_RECEIVER
= 513, asxCONTROL_AES18_BLOCK_GENERATOR = 514, asxCONTROL_-
CHANNEL_MODE = 515,
asxCONTROL_BIT_STREAM = 516, asxCONTROL_SAMPLE_CLOCK = 517,
asxCONTROL_MICROPHONE = 518, asxCONTROL_PARAMETRIC_EQ =
519,
asxCONTROL_COMPANDER = 520, asxCONTROL_COBRANET = 521, asxCONTROL_-
PLAYER = 522, asxCONTROL_RECORDER = 523,
asxCONTROL_GPIO = 524, asxCONTROL_RESERVED_525 = 525, asxCONTROL_-
RESERVED_526 = 526, asxCONTROL_RESERVED_527 = 527,
asxCONTROL_RESERVED_528 = 528, asxCONTROL_GENERIC = 529, asxCONTROL_-
TONEDETECTOR = 530, asxCONTROL_SILENCEDETECTOR = 531,
asxCONTROL_COBRANET_TRANSMITTER = 532, asxCONTROL_COBRANET_-
RECEIVER = 533, asxCONTROL_PAD = 534, asxCONTROL_SRC = 535,
asxCONTROL_BLOCK = 536, asxCONTROL_LAST_CONTROL = 537 }

```

Control type identifiers. The control types are used to differentiate control capabilities.

- enum `asxVOLUME_AUTOFADE` { `asxVOLUME_AUTOFADE_LOG`, `asxVOLUME_AUTOFADE_LINEAR` }
volume autofade profiles
- enum `asxMETER_TYPE` { `asxMETER_PEAK`, `asxMETER_RMS` }
Peak meter type to read.
- enum `asxCHANNELMODE` {
`asxCHANNELMODE_ILLEGAL` = 1000, `asxCHANNELMODE_NORMAL` = 1001,
`asxCHANNELMODE_SWAP` = 1002, `asxCHANNELMODE_STEREOTOLEFT` = 1003,
`asxCHANNELMODE_STEREOTORIGHT` = 1004, `asxCHANNELMODE_LEFTTOSTEREO` = 1005,
`asxCHANNELMODE_RIGHTTOSTEREO` = 1006 }
Channel mode settings.
- enum `asxADAPTERMODE` {
`asxADAPTERMODE_ILLEGAL` = 1100, `asxADAPTERMODE_4_PLAY` = 1101,
`asxADAPTERMODE_6_PLAY` = 1102, `asxADAPTERMODE_8_PLAY` = 1103,
`asxADAPTERMODE_9_PLAY` = 1104, `asxADAPTERMODE_12_PLAY` = 1105,
`asxADAPTERMODE_16_PLAY` = 1106, `asxADAPTERMODE_1_PLAY` = 1107,
`asxADAPTERMODE_MODE_1` = 1108, `asxADAPTERMODE_MODE_2` = 1109,
`asxADAPTERMODE_MODE_3` = 1110, `asxADAPTERMODE_MULTICHANNEL` = 1111,
`asxADAPTERMODE_MONO` = 1112, `asxADAPTERMODE_LOW_LATENCY` = 1113,
`asxADAPTERMODE_24_PLAY` = 1114, `asxADAPTERMODE_32_PLAY` = 1115 }

Adapter mode settings.

- enum `asxTUNERBAND` {
`asxTUNERBAND_AM` = 1200, `asxTUNERBAND_FM` = 1201, `asxTUNERBAND_TV` = 1202, `asxTUNERBAND_FM_STEREO` = 1203,
`asxTUNERBAND_AUX` = 1204, `asxTUNERBAND_TV_PAL_BG` = 1205, `asxTUNERBAND_TV_PAL_I` = 1206, `asxTUNERBAND_TV_PAL_DK` = 1207,
`asxTUNERBAND_TV_SECAM_L` = 1208, `asxTUNERBAND_DAB` = 1209
}

Tuner band settings.

- enum `asxTUNERDEEMPHASIS` { `asxTUNERDEEMPHASIS_50` = 1240, `asxTUNERDEEMPHASIS_75` = 1241, `asxTUNERDEEMPHASIS_NONE` = 1242 }

Tuner FM de-emphasis settings.

- enum `asxTUNERMODE` { `asxTUNERMODE_RSS` = 1250, `asxTUNERMODE_RSS_ENABLE` = 1251, `asxTUNERMODE_RSS_DISABLE` = 1252 }

Tuner mode settings.

- enum `asxTUNERPROGRAM` {
`asxTUNERPROGRAM_NONE` = 1260, `asxTUNERPROGRAM_1` = 1261, `asxTUNERPROGRAM_2` = 1262, `asxTUNERPROGRAM_3` = 1263,
`asxTUNERPROGRAM_4` = 1264, `asxTUNERPROGRAM_5` = 1265, `asxTUNERPROGRAM_6` = 1266, `asxTUNERPROGRAM_7` = 1267,
`asxTUNERPROGRAM_8` = 1268 }

Tuner program settings.

- enum `asxTUNERHDBLEND` { `asxTUNERHDBLEND_AUTO` = 1280, `asxTUNERHDBLEND_ANALOG` = 1281 }

Tuner HD Radio blend settings.

- enum `asxSAMPLE_CLOCK_SOURCE` {
`asxSAMPLE_CLOCK_SOURCE_ADAPTER` = 1400, `asxSAMPLE_CLOCK_SOURCE_AESEBUSYNC` = 1401, `asxSAMPLE_CLOCK_SOURCE_WORD` = 1402, `asxSAMPLE_CLOCK_SOURCE_WORD_HEADER` = 1403,
`asxSAMPLE_CLOCK_SOURCE_SMPTE` = 1404, `asxSAMPLE_CLOCK_SOURCE_NETWORK` = 1405, `asxSAMPLE_CLOCK_SOURCE_AESEBUAUTO` = 1406,
`asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT1` = 1407,
`asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT2` = 1408, `asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT3` = 1409, `asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT4` = 1410, `asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT5` = 1411,
`asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT6` = 1412, `asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT7` = 1413, `asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT8` = 1414, `asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT9` = 1415,

```

asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT10 = 1416, asxSAMPLE_-
CLOCK_SOURCE_AESEBUINPUT11 = 1417, asxSAMPLE_CLOCK_SOURCE_-
AESEBUINPUT12 = 1418, asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT13
= 1419,
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT14 = 1420, asxSAMPLE_-
CLOCK_SOURCE_AESEBUINPUT15 = 1421, asxSAMPLE_CLOCK_SOURCE_-
AESEBUINPUT16 = 1422, asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT17
= 1423,
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT18 = 1424, asxSAMPLE_-
CLOCK_SOURCE_AESEBUINPUT19 = 1425, asxSAMPLE_CLOCK_SOURCE_-
AESEBUINPUT20 = 1426, asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT21
= 1427,
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT22 = 1428, asxSAMPLE_-
CLOCK_SOURCE_AESEBUINPUT23 = 1429, asxSAMPLE_CLOCK_SOURCE_-
AESEBUINPUT24 = 1430, asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT25
= 1431,
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT26 = 1432, asxSAMPLE_-
CLOCK_SOURCE_AESEBUINPUT27 = 1433, asxSAMPLE_CLOCK_SOURCE_-
AESEBUINPUT28 = 1434, asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT29
= 1435,
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT30 = 1436, asxSAMPLE_-
CLOCK_SOURCE_AESEBUINPUT31 = 1437, asxSAMPLE_CLOCK_SOURCE_-
AESEBUINPUT32 = 1438, asxSAMPLE_CLOCK_SOURCE_LOCAL = 1439,
asxSAMPLE_CLOCK_SOURCE_PREV_MODULE = 1440, asxSAMPLE_CLOCK_-
SOURCE_UNDEFINED = 1441, asxSAMPLE_CLOCK_SOURCE_LIVEWIRE
= 1442, asxSAMPLE_CLOCK_SOURCE_BLULINK = 1443 }

```

Sample clock source options.

- enum `asxAESEBU_FORMAT` { `asxAESEBU_FORMAT_AESEBU` = 1450, `asxAESEBU_FORMAT_SPDIF` = 1451, `asxAESEBU_FORMAT_UNDEFINED` = 1452 }

Digital mode settings.

- enum `asxEQBANDTYPE` {
`asxEQBANDTYPE_BYPASS` = 1460, `asxEQBANDTYPE_LOWSHELF` = 1461,
`asxEQBANDTYPE_HIGHSHELF` = 1462, `asxEQBANDTYPE_EQUALIZER`
= 1463,
`asxEQBANDTYPE_LOWPASS` = 1464, `asxEQBANDTYPE_HIGHPASS` = 1465,
`asxEQBANDTYPE_BANDPASS` = 1466, `asxEQBANDTYPE_BANDSTOP` =
1467 }

Parametric equalizer band type settings.

- enum `asxCOBANET_MODE` { `asxCOBANET_MODE_NETWORK` = 1470,
`asxCOBANET_MODE_TETHERED` = 1471 }

Cobranet mode settings (deprecated!)

- enum `asxADPROPENUM_MODE` { `asxADPROPENUM_MODE_PROPERTIES`
= 1480, `asxADPROPENUM_MODE_SETTINGS` = 1481 }

Adapter property enumerate mode settings.

- enum `asxADPROPENUM_SSX2` { `asxADPROPENUM_SSX2_OFF` = 1488, `asxADPROPENUM_SSX2_ON` = 1489 }

Adapter property SSX2 enumerate settings.

- enum `asxCOBANET_LATENCY` { `asxCOBANET_LATENCY_133ms` = 1490, `asxCOBANET_LATENCY_266ms` = 1491, `asxCOBANET_LATENCY_533ms` = 1492 }

Cobranet latency settings.

- enum `asxFILE_FORMAT` { `asxFILE_FORMAT_WAV` = 10000, `asxFILE_FORMAT_RAW` = 10001 }

File Formats.

- enum `asxFILE_MODE` { `asxFILE_MODE_CREATE` = 10100, `asxFILE_MODE_APPEND` = 10101 }

File Mode.

- enum `asxAUDIO_FORMAT` {
`asxAUDIO_FORMAT_PCM8` = 10200, `asxAUDIO_FORMAT_PCM16` = 10201,
`asxAUDIO_FORMAT_PCM24` = 10202, `asxAUDIO_FORMAT_PCM32` = 10203,
`asxAUDIO_FORMAT_PCM32_FLOAT` = 10204, `asxAUDIO_FORMAT_MPEG_L2` = 10205, `asxAUDIO_FORMAT_MPEG_L3` = 10206, `asxAUDIO_FORMAT_MPEG_AACPLUS` = 10207,
`asxAUDIO_FORMAT_DOLBY_AC2` = 10208, `asxAUDIO_FORMAT_PCM20` = 10209, `asxAUDIO_FORMAT_NONE` = 10210 }

Audio Formats.

- enum `asxRECORD_MODE` {
`asxRECORD_MODE_STEREO` = 10300, `asxRECORD_MODE_JOINT_STEREO` = 10301, `asxRECORD_MODE_DUAL_MONO` = 10302, `asxRECORD_MODE_MONO` = 10303,
`asxRECORD_MODE_DONT_CARE` = 10304 }

Record Mode.

- enum `asxPLAYER_STATE` {
`asxPLAYER_INIT` = 10400, `asxPLAYER_OPEN` = 10401, `asxPLAYER_PREFILL` = 10402, `asxPLAYER_RUNNING` = 10403,
`asxPLAYER_PAUSED` = 10404, `asxPLAYER_DONE` = 10405, `asxPLAYER_DESTROY` = 10406 }

Player States.

- enum `asxRECORDER_STATE` {
`asxRECORDER_INIT` = 10500, `asxRECORDER_OPEN` = 10501, `asxRECORDER_RUNNING` = 10502, `asxRECORDER_PAUSED` = 10503,

`asxRECORDER_DONE = 10504, asxRECORDER_DESTROY = 10505 }`

Recorder States.

- `enum asxTIMESCALE {`
`asxTIMESCALE_INVALID = 10600, asxTIMESCALE_BYTES = 10601, asxTIMESCALE_-`
`MILLISECONDS = 10602, asxTIMESCALE_SAMPLES = 10603,`
`asxTIMESCALE_BYTES_REMAINING = 10604, asxTIMESCALE_MILLISECONDS_-`
`REMAINING = 10605, asxTIMESCALE_SAMPLES_REMAINING = 10606`
`}`

TimeScale type identifiers.

- `enum asxSAMPLE_RATE {`
`asxSAMPLE_RATE_8000 = 10700, asxSAMPLE_RATE_11025 = 10701, asxSAMPLE_-`
`RATE_16000 = 10702, asxSAMPLE_RATE_22050 = 10703,`
`asxSAMPLE_RATE_24000 = 10704, asxSAMPLE_RATE_32000 = 10705, asxSAMPLE_-`
`RATE_44100 = 10706, asxSAMPLE_RATE_48000 = 10707,`
`asxSAMPLE_RATE_64000 = 10708, asxSAMPLE_RATE_88200 = 10709, asxSAMPLE_-`
`RATE_96000 = 10710, asxSAMPLE_RATE_192000 = 10711,`
`asxSAMPLE_RATE_12000 = 10712, asxSAMPLE_RATE_176400 = 10713, asxSAMPLE_-`
`RATE_UNDEFINED = 10799 }`

Sample rate options.

- `enum asxMSG_LOGGING {`
`asxMSG_LOGGING_DISABLE = 10800, asxMSG_LOGGING_ERROR = 10801,`
`asxMSG_LOGGING_WARNING = 10802, asxMSG_LOGGING_NOTICE =`
`10803,`
`asxMSG_LOGGING_INFO = 10804, asxMSG_LOGGING_DEBUG = 10805,`
`asxMSG_LOGGING_VERBOSE = 10806 }`

Error logging control. Uses DbgView under Windows to log messages.

- `enum asxPLAYER_FLAGS { asxPLAYER_FILE_COMPLETE = 1, asxPLAYER_-`
`FILELIST_COMPLETE = 2, asxPLAYER_FILE_START = 4 }`

Player callback flags that form a bitmask. ie they are numbered 1,2,4,8 etc.

- `enum asxTUNER_STATUS {`
`asxTUNER_STATUS_VIDEO_VALID = 1, asxTUNER_STATUS_VIDEO_COLOR_-`
`PRESENT = 2, asxTUNER_STATUS_VIDEO_IS_60HZ = 4, asxTUNER_STATUS_-`
`VIDEO_HORZ_SYNC_MISSING = 8,`
`asxTUNER_STATUS_PLL_LOCKED = 16, asxTUNER_STATUS_FM_STEREO`
`= 32, asxTUNER_STATUS_DIGITAL = 64, asxTUNER_STATUS_MULTIPROGRAM`
`= 128,`
`asxTUNER_STATUS_FIRMWARE_LOADING = 256 }`

Tuner status bitfields. Not translatable to strings.

- enum `asxTUNER_RDS_TYPE` { `asxTUNER_RDS_TYPE_RDS` = 0, `asxTUNER_RDS_TYPE_RBDS` = 1 }

Tuner PSD/PAD/RDS/RBDS type. Not translatable to strings.

- enum `asxAESEBU_STATUS` {
`asxAESEBU_ERROR` = 0x01, `asxAESEBU_ERROR_NOT_LOCKED` = 0x02,
`asxAESEBU_ERROR_POOR_QUALITY` = 0x04, `asxAESEBU_ERROR_PARITY_ERROR` = 0x08,
`asxAESEBU_ERROR_BIPHASE_VIOLATION` = 0x10, `asxAESEBU_ERROR_VALIDITY` = 0x20, `asxAESEBU_ERROR_CHANNELSTATUS_CRC` = 0x40
}

AESEBU status bitfields. Not translatable to strings.

- enum `asxCOMPANDER_INDEX` { `asxCOMPANDER_INDEX_NOISEGATE` = 0, `asxCOMPANDER_INDEX_COMPANDER` = 1 }

Compander control indicies.

- enum `asxHANDLE_TYPE` {
`asxHANDLE_INVALID`, `asxHANDLE_SYSTEM`, `asxHANDLE_ADAPTER`,
`asxHANDLE_MIXER`,
`asxHANDLE_NODE`, `asxHANDLE_CONTROL`, `asxHANDLE_LAST` }

Handle type enums returned from `ASX_Handle_GetType()`.

- enum `asxUCONTROL_PTYPE` {
`asxPARAM_TYPE_NONE` = 0, `asxPARAM_TYPE_INTEGER` = 3, `asxPARAM_TYPE_FLOAT` = 4, `asxPARAM_TYPE_DOUBLE` = 5,
`asxPARAM_TYPE_STRING` = 6, `asxPARAM_TYPE_IP4_ADDRESS` = 8, `asxPARAM_TYPE_IP6_ADDRESS` = 9, `asxPARAM_TYPE_MAC_ADDRESS` = 10,
`asxPARAM_TYPE_BOOLEAN` = 11 }

Universal control parameter types.

- enum `asxUCONTROL_RTYPE` {
`asxPARAM_RANGE_NONE` = 0, `asxPARAM_RANGE_STEPPED_INTEGER` = 1, `asxPARAM_RANGE_STEPPED_FLOAT` = 2, `asxPARAM_RANGE_ENUMERATED_INTEGER` = 3,
`asxPARAM_RANGE_ENUMERATED_FLOAT` = 4, `asxPARAM_RANGE_ENUMERATED` = 5, `asxPARAM_RANGE_STRING_LENGTH` = 6, `asxPARAM_RANGE_NUMBER_OF_BITS` = 7 }

Universal control range types.

- enum `asxUCONTROL_PFLAGS` { `asxPARAM_FLAG_WRITEABLE` = 1, `asxPARAM_FLAG_READABLE` = 2, `asxPARAM_FLAG_VOLATILE` = 4 }

Universal control flags.

- enum `asxCOBNET_IFSTATUS` { `asxCOBNET_IFSTATUS_LINK_ESTABLISHED` = 0x01, `asxCOBNET_IFSTATUS_FULL_DUPLEX` = 0x02, `asxCOBNET_IFSTATUS_ACTIVE_CONNECTION` = 0x04 }

Cobranet If status bitfields. Not translatable to strings.

- enum `asxADAPTER_PROPERTY` {
`asxADAPTER_PROPERTY_ERRATA_1` = 1, `asxADAPTER_PROPERTY_SGX2_SETTING` = 2, `asxADAPTER_PROPERTY_SYNC_HEADER_CONNECTIONS` = 3, `asxADAPTER_PROPERTY_SUPPORT_SGX2` = 4,
`asxADAPTER_PROPERTY_SUPPORTS_FW_UPDATE` = 5, `asxADAPTER_PROPERTY_FIRMWARE_ID` = 6, `asxADAPTER_PROPERTY_SUPPORTS_SNMP` = 7 }

Properties for use with `ASX_Adapter_ReadProperty` and `ASX_Adapter_WriteProperty`.

Functions

- `ASX32_API int ASX_System_SupportsSubSystem` (const int `asxSystemType`)
Query ASX library for subsystem support.
- `ASX32_API ASX_ERROR ASX_System_Create` (const int `asxSystemType`, `ASX_HANDLE` *`phSystem`)
Create a complete ASX system.
- `ASX32_API ASX_ERROR ASX_System_CreateSubSystem` (const int `asxSystemType`, `ASX_HANDLE` *`pio_hSystem`)
Creates an ASX sub system and adds it to the existing system, if any.
- `ASX32_API ASX_ERROR ASX_System_SetHostNetworkInterface` (const char *`szInterface`)
Set the interface ASX should use when communicating with network devices.
- `ASX32_API ASX_ERROR ASX_System_SetAvdeccInterface` (const char *`szInterface`)
Set the interface ASX should use when initializing the AVDECC controller.
- `ASX32_API ASX_ERROR ASX_System_Delete` (`ASX_HANDLE` `hSystem`)
Delete a complete ASX system.
- `ASX32_API ASX_ERROR ASX_System_RegisterErrorCallback` (`ASX_HANDLE` `hSystem`, `ASX_ERROR_CALLBACK` *`pCallback`, void *`pUser1`, void *`pUser2`)
Register a callback function that should be called when an error is detected.
- `ASX32_API ASX_ERROR ASX_System_GetName` (`ASX_HANDLE` `hSystem`, char *`pszName`, const int `nStringLength`, int *`pnRequiredLength`)

Gets the name of the ASX system.

- ASX32_API [ASX_ERROR](#) [ASX_System_GetVersion](#) ([ASX_HANDLE](#) hSystem, char *pszSystemVersion, const int nSystemVersionLength, int *pnRequiredSystemVersionLength, char *pszSubSystemVersion, const int nSubSystemVersionLength, int *pnRequiredSubSystemVersionLength)

Get ASX system version information.

- ASX32_API [ASX_ERROR](#) [ASX_System_GetAdapterCount](#) ([ASX_HANDLE](#) hSystem, int *pnCount)

Get the number of adapters.

- ASX32_API [ASX_ERROR](#) [ASX_System_GetAdapter](#) ([ASX_HANDLE](#) hSystem, const int nAdapter, [ASX_HANDLE](#) *p_hAdapter)

Get a handle to a specific adapter.

- ASX32_API [ASX_ERROR](#) [ASX_System_SetMessageLogging](#) ([ASX_HANDLE](#) hSystem, const enum [asxMSG_LOGGING](#) eLog)

Set the message logging level for ASX.

- ASX32_API [ASX_ERROR](#) [ASX_System_GetMessageLogging](#) ([ASX_HANDLE](#) hSystem, enum [asxMSG_LOGGING](#) *eLog)

Get the message logging level for ASX.

- ASX32_API [ASX_ERROR](#) [ASX_System_SetCobranetAutoassignParms](#) (const struct [asxCobranetIpAutoassignParameters](#) *pCAP)

Set the IP address range that will be used for assigning IP addresses to cobranet devices.

- ASX32_API [ASX_ERROR](#) [ASX_System_GetCobranetAutoassignParms](#) (struct [asxCobranetIpAutoassignParameters](#) *pCAP)

Get the IP address range that will be used for assigning IP addresses to cobranet devices.

- ASX32_API enum [asxHANDLE_TYPE](#) [ASX_Handle_GetType](#) ([ASX_HANDLE](#) hHandle)

Get the handle type.

- ASX32_API [ASX_ERROR](#) [ASX_Error_GetLast](#) ([ASX_HANDLE](#) hASXObject, [ASX_ERROR](#) *pnAsxErrorCode, int *pnAsxSubSystemErrorCode)

Get the last error.

- ASX32_API [ASX_ERROR](#) [ASX_Error_GetLastString](#) ([ASX_HANDLE](#) hASXObject, char *pszAsxErrorString, const int nAsxErrorStringLength, int *pnRequiredAsxErrorStringLength, char *pszAsxSubSystemErrorString, const int nAsxSubSystemErrorStringLength, int *pnRequiredAsxSubSystemErrorStringLength)

Get the last error string information.

- ASX32_API [ASX_ERROR](#) [ASX_Error_Clear](#) ([ASX_HANDLE](#) hASXObject)

Clears the last error.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_CheckSubSystems](#) ([ASX_HANDLE](#) hAdapter, unsigned int *pnSubSystemMask, unsigned int *pnSubSystemOkMask)

Returns the status of the various sub-systems that interface to the adapter.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetName](#) ([ASX_HANDLE](#) hAdapter, char *pszName, const int nStringLength, int *RequiredLength)

Gets the name of the adapter.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetIndex](#) ([ASX_HANDLE](#) hAdapter, int *pnIndex)

Gets an adapter's index.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetAVDECCController](#) ([ASX_HANDLE](#) hAdapter, void **controller_obj)

Gets an adapter's AVDECC Controller object.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetAVDECCSystem](#) ([ASX_HANDLE](#) hAdapter, void **sys)

This function returns the system object used by adapters of subsystem type 1722.1.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetSerialNumber](#) ([ASX_HANDLE](#) hAdapter, unsigned long *pdwSerialNumber)

Gets an adapter's serial number.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetHardwareRevision](#) ([ASX_HANDLE](#) hAdapter, char *pszRevision)

Gets an adapter's hardware revision.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetFirmwareRevision](#) ([ASX_HANDLE](#) hAdapter, char *pszRevision)

Gets an adapter's firmware revision.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetMacAddress](#) ([ASX_HANDLE](#) hAdapter, char *pszMAC)

Gets an adapter's ethernet MAC address.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetIpAddress](#) ([ASX_HANDLE](#) hAdapter, char *pszIP)

Gets an adapter's network IP address.

- ASX32_API [ASX_ERROR](#) [ASX_Adapter_GetDspUtilization](#) ([ASX_HANDLE](#) hAdapter, const int nDspIndex, int *pnDspUtilization)

Gets an adapter's DSP utilization.

- ASX32_API ASX_ERROR ASX_Adapter_GetMixer (ASX_HANDLE hAdapter, ASX_HANDLE *p_hMixer)

Gets a handle to an adapter's mixer.

- ASX32_API ASX_ERROR ASX_Adapter_EnumerateMode (ASX_HANDLE hAdapter, const int nIndex, enum asxADAPTERMODE *peMode, int *pnCount)

Enumerate each adapter mode option.

- ASX32_API ASX_ERROR ASX_Adapter_GetMode (ASX_HANDLE hAdapter, enum asxADAPTERMODE *peMode)

Get the current adapter mode.

- ASX32_API ASX_ERROR ASX_Adapter_SetMode (ASX_HANDLE hAdapter, const enum asxADAPTERMODE eMode)

Set the current adapter mode.

- ASX32_API ASX_ERROR ASX_Adapter_EnumerateProperty (ASX_HANDLE hAdapter, const int nIndex, const enum asxADPROPENUM_MODE eMode, const int nSubIndex, unsigned long *pdwSetting)

Enumerates adapter properties and settings.

- ASX32_API ASX_ERROR ASX_Adapter_ReadProperty (ASX_HANDLE hAdapter, const int nIndex, unsigned short *pwParm1, unsigned short *pwParm2)

Read an adapter's property value.

- ASX32_API ASX_ERROR ASX_Adapter_WriteProperty (ASX_HANDLE hAdapter, const int nIndex, const unsigned short wParm1, const unsigned short wParm2)

Write an adapter property value.

- ASX32_API ASX_ERROR ASX_Adapter_WriteNvMem (ASX_HANDLE hAdapter, const int nAddress, const unsigned char cValue)

Write a byte to the non-volatile memory.

- ASX32_API ASX_ERROR ASX_Adapter_ReadNvMem (ASX_HANDLE hAdapter, const int nAddress, unsigned char *pcValue)

Read a byte from the non-volatile memory.

- ASX32_API ASX_ERROR ASX_Adapter_GetNvMemSizeInBytes (ASX_HANDLE hAdapter, int *pnCount)

Get the number of bytes in the adapter's non-volatile memory.

- ASX32_API ASX_ERROR ASX_Mixer_ResetControls (ASX_HANDLE hMixer)

This function sets all the controls in the mixer to a known state.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetSourceNodeCount](#) ([ASX_HANDLE](#) hMixer, int *pnCount)

This function returns the number of source nodes in the mixer.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetSourceNode](#) ([ASX_HANDLE](#) hMixer, const int nSourceNode, [ASX_HANDLE](#) *p_hNode)

This function gets the handle of a particular source node.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetDestinationNodeCount](#) ([ASX_HANDLE](#) hMixer, int *pnCount)

This function returns the number of destination nodes in the mixer.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetDestinationNode](#) ([ASX_HANDLE](#) hMixer, const int nDestinationNode, [ASX_HANDLE](#) *p_hNode)

This function gets the handle of the specified destination node.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetNodeByType](#) ([ASX_HANDLE](#) hMixer, const enum [asxNODE](#) eType, const int nIndex, [ASX_HANDLE](#) *p_hNode)

Get a node by type.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetNodeTypeCount](#) ([ASX_HANDLE](#) hMixer, const enum [asxNODE](#) eType, int *pnCount)

Get the number of nodes of the specified type.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetControlCount](#) ([ASX_HANDLE](#) hMixer, int *pnControls)

This function returns the total number of controls in the mixer.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetControl](#) ([ASX_HANDLE](#) hMixer, const int nControl, [ASX_HANDLE](#) *p_hControlBase)

Given a control index, this function returns a handle to the specified control.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetControlByNode](#) (const [ASX_HANDLE](#) hMixer, const [ASX_HANDLE](#) hSourceNode, const [ASX_HANDLE](#) hDestinationNode, const enum [asxCONTROL](#) eControlType, [ASX_HANDLE](#) *p_hControlBase)

Given source and destination node handles as well as the control type, return the specified control.

- ASX32_API [ASX_ERROR](#) [ASX_Mixer_GetControlByNodeTypeAndIndex](#) ([ASX_HANDLE](#) hMixer, const enum [asxNODE](#) nSourceNodeType, const int nSourceIndex, const enum [asxNODE](#) nDestinationNodeType, const int nDestinationIndex, const enum [asxCONTROL](#) eControlType, [ASX_HANDLE](#) *p_hControlBase)

Given source and destination node specifications as well as the control type, return the specified control.

- ASX32_API [ASX_ERROR ASX_Mixer_GetBlockControlByNodeTypeAndIndex](#) ([ASX_HANDLE](#) hMixer, const enum [asxNODE](#) nSourceNodeType, const int nSourceIndex, const enum [asxNODE](#) nDestinationNodeType, const int nDestinationIndex, const char *pszBlockName, [ASX_HANDLE](#) *p_hControlBase)
Given source and destination node specifications as well as the block control name, return the specified control.
- ASX32_API [ASX_ERROR ASX_Node_GetType](#) ([ASX_HANDLE](#) hNode, enum [asxNODE](#) *peType)
Returns the node type of the given node.
- ASX32_API [ASX_ERROR ASX_Node_GetIndex](#) ([ASX_HANDLE](#) hNode, int *pnIndex)
Returns the index of the given node.
- ASX32_API [ASX_ERROR ASX_Node_GetLocation](#) ([ASX_HANDLE](#) hNode, int *pnModuleSlot, int *pnNodeIndexOnSlot, char *pszModuleName, const int nStringLength)
Returns the location of the given node in terms of module slots and position on the module that contains the node.
- ASX32_API [ASX_ERROR ASX_Node_GetSubSystem](#) ([ASX_HANDLE](#) hNode, int *p_nSubSystem)
Returns the sub system handle of the given node.
- ASX32_API [ASX_ERROR ASX_Node_GetName](#) ([ASX_HANDLE](#) hNode, char *pszNodeName, const int nStringLength)
Get the name of the node.
- ASX32_API [ASX_ERROR ASX_Mixer_GetNodeType](#) ([ASX_HANDLE](#) hNode, enum [asxNODE](#) *peType)
- ASX32_API [ASX_ERROR ASX_Mixer_GetNodeIndex](#) ([ASX_HANDLE](#) hNode, int *pnIndex)
- ASX32_API [ASX_ERROR ASX_Control_GetType](#) ([ASX_HANDLE](#) hControl, enum [asxCONTROL](#) *peControl)
Generic control function to get the type of a control.
- ASX32_API [ASX_ERROR ASX_Control_GetSourceNode](#) ([ASX_HANDLE](#) hControl, [ASX_HANDLE](#) *p_hNode)
Generic control function to get the source node of a control.
- ASX32_API [ASX_ERROR ASX_Control_GetDestinationNode](#) ([ASX_HANDLE](#) hControl, [ASX_HANDLE](#) *p_hNode)
Generic control function to get the destination node of a control.
- ASX32_API [ASX_ERROR ASX_Control_GetHpiControl](#) ([ASX_HANDLE](#) hControl, void **pphHpiSubSys, unsigned int *phHpiControl)

Tunnel through ASX to get HPI control parameters (NOT IMPLEMENTED YET).

- ASX32_API [ASX_ERROR](#) [ASX_Control_GetSubSystem](#) ([ASX_HANDLE](#) hControl, int *p_nSubSystem)
Returns the sub system handle of the given control.
- ASX32_API [ASX_ERROR](#) [ASX_Player_Open](#) ([ASX_HANDLE](#) hPlayer, const char *pszFile)
Open a file for playback.
- ASX32_API [ASX_ERROR](#) [ASX_Player_Format_GetString](#) ([ASX_HANDLE](#) hPlayer, char **pszFormat)
Get the format of the currently opened file as a string.
- ASX32_API [ASX_ERROR](#) [ASX_Player_Format_GetDetails](#) ([ASX_HANDLE](#) hPlayer, enum [asxAUDIO_FORMAT](#) *peFormat, int *pnChannnels, int *pnSampleRate, int *pnBitRate)
Get the format of the currently opened file as individual variables.
- ASX32_API [ASX_ERROR](#) [ASX_Player_PreLoad](#) ([ASX_HANDLE](#) hPlayer, const enum [asxTIMESCALE](#) nType, const unsigned long lPosition)
Preloads playback buffers from the given position, ready for playback.
- ASX32_API [ASX_ERROR](#) [ASX_Player_Start](#) ([ASX_HANDLE](#) hPlayer)
Start playback of a previously opened (and optionally pre-loaded) file.
- ASX32_API [ASX_ERROR](#) [ASX_Player_Pause](#) ([ASX_HANDLE](#) hPlayer)
Pause playback of the currently playing file.
- ASX32_API [ASX_ERROR](#) [ASX_Player_Stop](#) ([ASX_HANDLE](#) hPlayer)
Stops playback of the currently playing file.
- ASX32_API [ASX_ERROR](#) [ASX_Player_Wait](#) ([ASX_HANDLE](#) hPlayer)
Wait for the current file to finish.
- ASX32_API [ASX_ERROR](#) [ASX_Player_Close](#) ([ASX_HANDLE](#) hPlayer)
Close the current playback file.
- ASX32_API [ASX_ERROR](#) [ASX_Player_GetPosition](#) ([ASX_HANDLE](#) hPlayer, const enum [asxTIMESCALE](#) nType, unsigned long *plPosition)
Get the current playback position as the offset in bytes, samples or milliseconds from the beginning or end of the file depending on the timescale code used.
- ASX32_API [ASX_ERROR](#) [ASX_Player_SetPosition](#) ([ASX_HANDLE](#) hPlayer, const enum [asxTIMESCALE](#) nType, const unsigned long lPosition)
Sets the playback to the given position.

- ASX32_API [ASX_ERROR](#) [ASX_Player_GetState](#) ([ASX_HANDLE](#) hPlayer, enum [asxPLAYER_STATE](#) *pnState)
Get the current playback state.
- ASX32_API [ASX_ERROR](#) [ASX_Player_SetTimeScale](#) ([ASX_HANDLE](#) hPlayer, const float fTimeScale)
Set the playback timescale.
- ASX32_API [ASX_ERROR](#) [ASX_Player_GetTimeScale](#) ([ASX_HANDLE](#) hPlayer, float *pfTimeScale)
Get the playback timescale.
- ASX32_API [ASX_ERROR](#) [ASX_Player_GetFilename](#) ([ASX_HANDLE](#) hPlayer, char *pszFilename, const int nStringLength, int *pnRequiredLength)
Get the current filename, if any.
- ASX32_API [ASX_ERROR](#) [ASX_Player_SetLoopMode](#) ([ASX_HANDLE](#) hPlayer, const int nLooping)
Set the player to loop or single play mode.
- ASX32_API [ASX_ERROR](#) [ASX_Player_GetLoopMode](#) ([ASX_HANDLE](#) hPlayer, int *pnLooping)
Get the current player loop mode.
- ASX32_API [ASX_ERROR](#) [ASX_Player_OpenPlaylist](#) ([ASX_HANDLE](#) hPlayer, const char **pszFileList, const unsigned int nFiles)
Open a list of files for playback.
- ASX32_API [ASX_ERROR](#) [ASX_Player_PlaylistStatus](#) ([ASX_HANDLE](#) hPlayer, unsigned int *nTotalFileCount, int *nCurrentFile, char **szCurrentFilename, unsigned int *nTotalTime_ms, unsigned int *nCurrentTime_ms)
Returns playlist status.
- ASX32_API [ASX_ERROR](#) [ASX_Player_RegisterCallback](#) ([ASX_HANDLE](#) hPlayer, [ASX_PLAYER_CALLBACK](#) *pCallback, const enum [asxPLAYER_FLAGS](#) flags, void *pUser1)
Register a callback function that should be called when playback has completed.
- ASX32_API [ASX_ERROR](#) [ASX_Player_PlaylistWait](#) ([ASX_HANDLE](#) hPlayer)
Wait for the playlist to finish.
- ASX32_API [ASX_ERROR](#) [ASX_Recorder_Open](#) ([ASX_HANDLE](#) hRecorder, const char *pszFile, const enum [asxFILE_FORMAT](#) nFileType, const enum [asxFILE_MODE](#) nFileMode, const int nChannels, const enum [asxAUDIO_FORMAT](#) nFormat, const long lSampleRate, const long lBitrate, const enum [asxRECORD_MODE](#) nMode)

Opens the recorder using the specified format.

- ASX32_API [ASX_ERROR](#) [ASX_Recorder_Start](#) ([ASX_HANDLE](#) hRecorder)

Starts the recording.

- ASX32_API [ASX_ERROR](#) [ASX_Recorder_Stop](#) ([ASX_HANDLE](#) hRecorder)

Stops the recording.

- ASX32_API [ASX_ERROR](#) [ASX_Recorder_Pause](#) ([ASX_HANDLE](#) hRecorder)

Pauses the recording.

- ASX32_API [ASX_ERROR](#) [ASX_Recorder_Close](#) ([ASX_HANDLE](#) hRecorder)

Closes the recording file.

- ASX32_API [ASX_ERROR](#) [ASX_Recorder_GetPosition](#) ([ASX_HANDLE](#) hRecorder, const enum [asxTIMESCALE](#) nType, unsigned long *plPosition)

Gets the current record position.

- ASX32_API [ASX_ERROR](#) [ASX_Recorder_GetState](#) ([ASX_HANDLE](#) hRecorder, enum [asxRECORDER_STATE](#) *peState)

Get the current record state.

- ASX32_API [ASX_ERROR](#) [ASX_Recorder_GetFilename](#) ([ASX_HANDLE](#) hRecorder, char *pszFilename, const int nStringLength, int *pnRequiredLength)

Get the current filename, if any.

- ASX32_API [ASX_ERROR](#) [ASX_Recorder_EnumerateFormat](#) ([ASX_HANDLE](#) hRecorder, const int nIndex, enum [asxAUDIO_FORMAT](#) *peFormat, int *pnCount)

Enumerates supported recorder formats.

- ASX32_API [ASX_ERROR](#) [ASX_Meter_GetChannels](#) ([ASX_HANDLE](#) hMeter, int *pnChannels)

Returns the number of channels this peak meter has.

- ASX32_API [ASX_ERROR](#) [ASX_Meter_GetPeak](#) ([ASX_HANDLE](#) hMeter, float *fdB, const int nChannels)

Returns the peak meter reading for the given meter control.

- ASX32_API [ASX_ERROR](#) [ASX_Meter_GetRMS](#) ([ASX_HANDLE](#) hMeter, float *fdB, const int nChannels)

Returns the RMS meter reading for the given meter control.

- ASX32_API [ASX_ERROR](#) [ASX_Meter_SetBallistics](#) ([ASX_HANDLE](#) hMeter, const enum [asxMETER_TYPE](#) nMeterType, const float fAttackTimeMs, const float fDecayTimeMs)

Set the meter ballistics.

- ASX32_API [ASX_ERROR](#) [ASX_Meter_GetBallistics](#) ([ASX_HANDLE](#) hMeter, const enum [asxMETER_TYPE](#) nMeterType, float *fAttackTimeMs, float *fDecayTimeMs)

Get meter ballistics.

- ASX32_API [ASX_ERROR](#) [ASX_Volume_GetChannels](#) ([ASX_HANDLE](#) hVolume, int *pnChannels)

Returns the number of channels this volume control has.

- ASX32_API [ASX_ERROR](#) [ASX_Volume_SetMute](#) ([ASX_HANDLE](#) hVolume, int *mute, const int nChannels)

Sets mute for this volume control.

- ASX32_API [ASX_ERROR](#) [ASX_Volume_GetMute](#) ([ASX_HANDLE](#) hVolume, int *mute, const int nChannels)

Returns the mute setting for this volume control.

- ASX32_API [ASX_ERROR](#) [ASX_Volume_SetGain](#) ([ASX_HANDLE](#) hVolume, float *fSetdB, const int nChannels)

Set volume.

- ASX32_API [ASX_ERROR](#) [ASX_Volume_GetGain](#) ([ASX_HANDLE](#) hVolume, float *fdB, const int nChannels)

Get volume.

- ASX32_API [ASX_ERROR](#) [ASX_Volume_GetRange](#) ([ASX_HANDLE](#) hVolume, float *fMinGain, float *fMaxGain, float *fGainStep)

Get that range of volume settings available.

- ASX32_API [ASX_ERROR](#) [ASX_Volume_SetAutofade](#) ([ASX_HANDLE](#) hVolume, const float *fSetdB, const int nChannels, const [ASX_TIME](#) nDuration, const enum [asxVOLUME_AUTOFADE](#) eProfile)

Set an autofade operation.

- ASX32_API [ASX_ERROR](#) [ASX_Level_Set](#) ([ASX_HANDLE](#) hLevel, const float fGain)

Set the analog input or output level (sometimes called trim).

- ASX32_API [ASX_ERROR](#) [ASX_Level_Get](#) ([ASX_HANDLE](#) hLevel, float *fGain)

Get the analog input or output level (sometimes called trim).

- ASX32_API [ASX_ERROR ASX_Level_GetRange](#) ([ASX_HANDLE](#) hLevel, float *fMinGain, float *fMaxGain, float *fGainStep)
Get that range of level settings available.
- ASX32_API [ASX_ERROR ASX_Multiplexer_Enumerate](#) ([ASX_HANDLE](#) hMux, const int nIndex, enum [asxNODE](#) *peNode, int *pnNodeIndex, int *pnCount)
Enumerate each multiplexer option.
- ASX32_API [ASX_ERROR ASX_Multiplexer_Get](#) ([ASX_HANDLE](#) hMux, enum [asxNODE](#) *peNode, int *pnNodeIndex)
Get the current multiplexer setting.
- ASX32_API [ASX_ERROR ASX_Multiplexer_Set](#) ([ASX_HANDLE](#) hMux, const enum [asxNODE](#) eNode, const int nIndex)
Set the multiplexer.
- ASX32_API [ASX_ERROR ASX_ChannelMode_Enumerate](#) ([ASX_HANDLE](#) hMode, const int nIndex, enum [asxCHANNELMODE](#) *peMode, int *pnCount)

Enumerate each channel mode option.
- ASX32_API [ASX_ERROR ASX_ChannelMode_Get](#) ([ASX_HANDLE](#) hMode, enum [asxCHANNELMODE](#) *peMode)
Get the current channel mode.
- ASX32_API [ASX_ERROR ASX_ChannelMode_Set](#) ([ASX_HANDLE](#) hMode, const enum [asxCHANNELMODE](#) eMode)
Set the current channel mode.
- ASX32_API [ASX_ERROR ASX_Tuner_EnumerateBand](#) ([ASX_HANDLE](#) hTuner, const int nIndex, enum [asxTUNERBAND](#) *peBand, int *pnCount)
Enumerate each tuner band option.
- ASX32_API [ASX_ERROR ASX_Tuner_GetBand](#) ([ASX_HANDLE](#) hTuner, enum [asxTUNERBAND](#) *peBand)
Get the tuner band.
- ASX32_API [ASX_ERROR ASX_Tuner_SetBand](#) ([ASX_HANDLE](#) hTuner, const enum [asxTUNERBAND](#) eBand)
Set the tuner band.
- ASX32_API [ASX_ERROR ASX_Tuner_SetFrequency](#) ([ASX_HANDLE](#) hTuner, const unsigned long nFreq)
Set the tuner frequency.
- ASX32_API [ASX_ERROR ASX_Tuner_GetFrequency](#) ([ASX_HANDLE](#) hTuner, unsigned long *plFreq)

Get the tuner frequency.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetFrequencyRange](#) ([ASX_HANDLE](#) hTuner, const enum [asxTUNERBAND](#) eBand, unsigned long *plMin, unsigned long *plMax, unsigned long *plStep)

Get the tuner frequency range in Hz.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetGainRange](#) ([ASX_HANDLE](#) hTuner, float *fMin, float *fMax, float *fStep)

Get the tuner gain range (in dB).

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_SetGain](#) ([ASX_HANDLE](#) hTuner, const float fTunerGain)

Set the tuner gain.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetGain](#) ([ASX_HANDLE](#) hTuner, float *pfTunerGain)

Get the tuner gain.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetRFLevel](#) ([ASX_HANDLE](#) hTuner, float *nRFLevel)

Get the tuner RF level.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetRawRFLevel](#) ([ASX_HANDLE](#) hTuner, int *nRawRFLevel)

Get the Raw tuner RF level.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetStatus](#) ([ASX_HANDLE](#) hTuner, unsigned int *puErrorStatusMask, unsigned int *puErrorStatus)

Get the tuner status.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetMode](#) ([ASX_HANDLE](#) hTuner, const enum [asxTUNERMODE](#) eMode, enum [asxTUNERMODE](#) *peSetting)

Gets the tuner mode.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_SetMode](#) ([ASX_HANDLE](#) hTuner, const enum [asxTUNERMODE](#) eMode, const enum [asxTUNERMODE](#) eSetting)

Sets the tuner mode.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_EnumerateDeemphasis](#) ([ASX_HANDLE](#) hTuner, const int nIndex, enum [asxTUNERDEEMPHASIS](#) *peDeemphasis, int *pnCount)

Enumerates tuner de-emphasis options.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_SetDeemphasis](#) ([ASX_HANDLE](#) hTuner, const enum [asxTUNERDEEMPHASIS](#) eDeemphasis)

Set tuner de-emphasis.

- ASX32_API [ASX_ERROR ASX_Tuner_GetDeemphasis](#) (ASX_HANDLE hTuner, enum [asxTUNERDEEMPHASIS](#) *peDeemphasis)
Get tuner de-emphasis.
- ASX32_API [ASX_ERROR ASX_Tuner_EnumerateProgram](#) (ASX_HANDLE hTuner, const int nIndex, enum [asxTUNERPROGRAM](#) *peProgram, int *pnCount)
Enumerates tuner program options.
- ASX32_API [ASX_ERROR ASX_Tuner_SetProgram](#) (ASX_HANDLE hTuner, const enum [asxTUNERPROGRAM](#) eProgram)
Set tuner program.
- ASX32_API [ASX_ERROR ASX_Tuner_GetProgram](#) (ASX_HANDLE hTuner, enum [asxTUNERPROGRAM](#) *peProgram)
Get tuner program.
- ASX32_API [ASX_ERROR ASX_Tuner_GetHdRadioSignalQuality](#) (ASX_HANDLE hTuner, int *pnSignalQuality)
- ASX32_API [ASX_ERROR ASX_Tuner_GetDigitalSignalQuality](#) (ASX_HANDLE hTuner, int *pnSignalQuality)
Get digital signal quality.
- ASX32_API [ASX_ERROR ASX_Tuner_GetHdRadioSdkVersion](#) (ASX_HANDLE hTuner, char *szSdkVersion, const int nStringLength)
- ASX32_API [ASX_ERROR ASX_Tuner_GetHdRadioDspVersion](#) (ASX_HANDLE hTuner, char *szSdkVersion, const int nStringLength)
- ASX32_API [ASX_ERROR ASX_Tuner_GetFirmwareVersion](#) (ASX_HANDLE hTuner, char *szFirmwareVersion, const int nStringLength)
Get a Firmware version string.
- ASX32_API [ASX_ERROR ASX_Tuner_EnumerateHdBlend](#) (ASX_HANDLE hTuner, const int nIndex, enum [asxTUNERHDBLEND](#) *peBlend, int *pnCount)
Enumerates tuner blend options.
- ASX32_API [ASX_ERROR ASX_Tuner_SetHdBlend](#) (ASX_HANDLE hTuner, const enum [asxTUNERHDBLEND](#) nMode)
Set a HD Radio tuner to analog only or auto switch.
- ASX32_API [ASX_ERROR ASX_Tuner_GetHdBlend](#) (ASX_HANDLE hTuner, enum [asxTUNERHDBLEND](#) *pnMode)
Get a HD Radio tuner analog or digital blend.
- ASX32_API [ASX_ERROR ASX_Tuner_GetDabMultiplexName](#) (ASX_HANDLE hTuner, char *szMultiplexName, const int nSize)

Get a DAB Multiplex Name.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetDabMultiplexId](#) ([ASX_HANDLE](#) hTuner, unsigned long *dwMultiplexId)

Get a DAB Multiplex ID.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetDabAudioServiceCount](#) ([ASX_HANDLE](#) hTuner, int *pnIndex, int *pnCount)

Get Number of Dab Audio Services.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetDabAudioServiceName](#) ([ASX_HANDLE](#) hTuner, char *szAudioServiceName, const int nSize, const int nIndex)

Get a DAB Audio Service.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_SetDabAudioService](#) ([ASX_HANDLE](#) hTuner, const int nIndex)

Set a DAB Audio Service.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetDabServiceId](#) ([ASX_HANDLE](#) hTuner, unsigned long *dwServiceId)

Get a DAB Service ID.

- ASX32_API [ASX_ERROR](#) [ASX_Tuner_GetDabAudioInfo](#) ([ASX_HANDLE](#) hTuner, char *szAudioInfo, const int nSize)

Get a DAB audio information.

- ASX32_API [ASX_ERROR](#) [ASX_PAD_GetChannelName](#) ([ASX_HANDLE](#) hPAD, char *pszChannelName, const int nStringLength)

Get a Program Auxiliary Data channel name.

- ASX32_API [ASX_ERROR](#) [ASX_PAD_GetArtist](#) ([ASX_HANDLE](#) hPAD, char *pszArtist, const int nStringLength)

Get a Program Auxiliary Data artist.

- ASX32_API [ASX_ERROR](#) [ASX_PAD_GetTitle](#) ([ASX_HANDLE](#) hPAD, char *pszTitle, const int nStringLength)

Get a Program Auxiliary Data title.

- ASX32_API [ASX_ERROR](#) [ASX_PAD_GetComment](#) ([ASX_HANDLE](#) hPAD, char *pszComment, const int nStringLength)

Get a Program Auxiliary Data comment.

- ASX32_API [ASX_ERROR](#) [ASX_PAD_GetProgramType](#) ([ASX_HANDLE](#) hPAD, int *pnProgramType)

Get a Program Auxiliary Data program type (PTY).

- ASX32_API [ASX_ERROR ASX_PAD_GetProgramTypeString](#) ([ASX_HANDLE](#) hPAD, const enum [asxTUNER_RDS_TYPE](#) eType, const int nPTY, char *pszString, const int nStringLength)
Get a Program Auxiliary Data PTY string.
- ASX32_API [ASX_ERROR ASX_PAD_GetRdsPI](#) ([ASX_HANDLE](#) hPAD, int *uPI)
Get a Program Identification number.
- ASX32_API [ASX_ERROR ASX_SampleClock_EnumerateSampleRate](#) ([ASX_HANDLE](#) hSampleClock, const int nIndex, enum [asxSAMPLE_RATE](#) *peSampleRate, int *pnCount)
- ASX32_API [ASX_ERROR ASX_SampleClock_EnumerateLocalRate](#) ([ASX_HANDLE](#) hSampleClock, const int nIndex, enum [asxSAMPLE_RATE](#) *peSampleRate, int *pnCount)
Enumerate each sample clock rates for the local sample clock generator.
- ASX32_API [ASX_ERROR ASX_SampleClock_SetSampleRate](#) ([ASX_HANDLE](#) hSampleClock, const int nSampleRate)
- ASX32_API [ASX_ERROR ASX_SampleClock_SetLocalRate](#) ([ASX_HANDLE](#) hSampleClock, const int nSampleRate)
Set the sample rate for the local sample clock generator.
- ASX32_API [ASX_ERROR ASX_SampleClock_GetSampleRate](#) ([ASX_HANDLE](#) hSampleClock, int *pnSampleRate)
Get the adapter's sample rate.
- ASX32_API [ASX_ERROR ASX_SampleClock_GetLocalRate](#) ([ASX_HANDLE](#) hSampleClock, int *pnSampleRate)
Get the sample rate for the local sample clock generator.
- ASX32_API [ASX_ERROR ASX_SampleClock_EnumerateClockSource](#) ([ASX_HANDLE](#) hSampleClock, const int nIndex, enum [asxSAMPLE_CLOCK_SOURCE](#) *peClockSource, int *pnCount)
Enumerate each sample clock source option.
- ASX32_API [ASX_ERROR ASX_SampleClock_SetClockSource](#) ([ASX_HANDLE](#) hSampleClock, const enum [asxSAMPLE_CLOCK_SOURCE](#) eClockSource)
Set the sample clock source.
- ASX32_API [ASX_ERROR ASX_SampleClock_GetClockSource](#) ([ASX_HANDLE](#) hSampleClock, enum [asxSAMPLE_CLOCK_SOURCE](#) *peClockSource)
Get the sample clock source.
- ASX32_API [ASX_ERROR ASX_SampleClock_SetAutoSource](#) ([ASX_HANDLE](#) hSampleClock, const int nEnable)
Set the sample clock to automatically source its clock from a valid input.

- ASX32_API [ASX_ERROR ASX_SampleClock_GetAutoSource](#) (ASX_HANDLE hSampleClock, int *pnEnable)
Get the setting of the auto source property of the sample clock.
- ASX32_API [ASX_ERROR ASX_SampleClock_SetLocalRateLock](#) (ASX_HANDLE hSampleClock, const int nLock)
Lock the local sample clock to its current setting.
- ASX32_API [ASX_ERROR ASX_SampleClock_GetLocalRateLock](#) (ASX_HANDLE hSampleClock, int *pnLock)
Get the setting of the local sample clock lock.
- ASX32_API [ASX_ERROR ASX_AESEBUReceiver_GetErrorStatus](#) (ASX_HANDLE hAESEBURx, unsigned int *pdwErrorStatusMask, unsigned int *pdwErrorStatus)

Get the status of the AESEBU receiver.
- ASX32_API [ASX_ERROR ASX_AESEBUReceiver_GetSampleRate](#) (ASX_HANDLE hAESEBURx, unsigned int *pdwSampleRate)
Get the sample rate of the AESEBU receiver.
- ASX32_API [ASX_ERROR ASX_AESEBUReceiver_EnumerateFormat](#) (ASX_HANDLE hAESEBURx, const int nIndex, enum [asxAESEBU_FORMAT](#) *peAesebuFormat, int *pnCount)
Enumerate each AES3 receive format supported by the hardware.
- ASX32_API [ASX_ERROR ASX_AESEBUReceiver_SetFormat](#) (ASX_HANDLE hAESEBURx, const enum [asxAESEBU_FORMAT](#) eAesebuFormat)
Set the format of the AESEBU receiver.
- ASX32_API [ASX_ERROR ASX_AESEBUReceiver_GetFormat](#) (ASX_HANDLE hAESEBURx, enum [asxAESEBU_FORMAT](#) *peAesebuFormat)
Get the format of the AESEBU receiver.
- ASX32_API [ASX_ERROR ASX_AESEBUTransmitter_EnumerateFormat](#) (ASX_HANDLE hAESEBUTx, const int nIndex, enum [asxAESEBU_FORMAT](#) *peAesebuFormat, int *pnCount)
Enumerate each AES3 transmit format supported by the hardware.
- ASX32_API [ASX_ERROR ASX_AESEBUTransmitter_SetFormat](#) (ASX_HANDLE hAESEBUTx, const enum [asxAESEBU_FORMAT](#) eAesebuFormat)
Set the format of the AESEBU transmitter.
- ASX32_API [ASX_ERROR ASX_AESEBUTransmitter_GetFormat](#) (ASX_HANDLE hAESEBUTx, enum [asxAESEBU_FORMAT](#) *peAesebuFormat)
Get the format of the AESEBU transmitter.

- ASX32_API [ASX_ERROR](#) [ASX_GPIO_GetProperties](#) ([ASX_HANDLE](#) hGPIO, int *pnNumberOfInputBits, int *pnNumberOfOutputBits)
Get the properties of the GPIO control.
- ASX32_API [ASX_ERROR](#) [ASX_GPIO_InputGet](#) ([ASX_HANDLE](#) hGPIO, int *pnInputBits, const int nNumberOfBits)
Read the state of the GPIO opto inputs.
- ASX32_API [ASX_ERROR](#) [ASX_GPIO_OutputSet](#) ([ASX_HANDLE](#) hGPIO, int *pnOutputBits, const int nNumberOfBits)
Write to the GPIO relay outputs.
- ASX32_API [ASX_ERROR](#) [ASX_GPIO_OutputGet](#) ([ASX_HANDLE](#) hGPIO, int *pnOutputBits, const int nNumberOfBits)
Read the current GPIO relay output settings.
- ASX32_API [ASX_ERROR](#) [ASX_Vox_SetLevel](#) ([ASX_HANDLE](#) hVox, const float fSetLevel)
Set vox level.
- ASX32_API [ASX_ERROR](#) [ASX_Vox_GetLevel](#) ([ASX_HANDLE](#) hVox, float *fGetLevel)
Get vox level.
- ASX32_API [ASX_ERROR](#) [ASX_Vox_GetRange](#) ([ASX_HANDLE](#) hVox, float *fMinLevel, float *fMaxLevel, float *fLevelStep)
Get that range of vox settings available.
- ASX32_API [ASX_ERROR](#) [ASX_GetGenericControlName](#) ([ASX_HANDLE](#) hControl, char *pName)
Get the name of the control.
- ASX32_API [ASX_ERROR](#) [ASX_Mic_SetPhantomPower](#) ([ASX_HANDLE](#) hMic, const int nOnOff)
Turn the phantom power on or off.
- ASX32_API [ASX_ERROR](#) [ASX_Mic_GetPhantomPower](#) ([ASX_HANDLE](#) hMic, int *pOnOff)
Get the current state of the phantom power (on or off).
- ASX32_API [ASX_ERROR](#) [ASX_EQ_GetInfo](#) ([ASX_HANDLE](#) hParmEq, unsigned short *pwNumberOfBands, unsigned short *pwEnabled)
Gets information on the equalizer.
- ASX32_API [ASX_ERROR](#) [ASX_EQ_SetState](#) ([ASX_HANDLE](#) hParmEq, const unsigned short wOnOff)

Turns the equalizer on or off.

- ASX32_API [ASX_ERROR](#) [ASX_EQ_SetBand](#) ([ASX_HANDLE](#) hParmEq, const unsigned short wIndex, const enum [asxEQBANDTYPE](#) eType, const unsigned long dwFrequencyHz, const short nQ100, const short nGain0_01dB)

Sets the parameters for an equalizer band.

- ASX32_API [ASX_ERROR](#) [ASX_EQ_GetBand](#) ([ASX_HANDLE](#) hParmEq, const unsigned short wIndex, enum [asxEQBANDTYPE](#) *peType, unsigned long *pdwFrequencyHz, short *pnQ100, short *pnGain0_01dB)

Gets the parameters for an equalizer band.

- ASX32_API [ASX_ERROR](#) [ASX_Compander_Set](#) ([ASX_HANDLE](#) hCompander, const unsigned short wAttack, const unsigned short wDecay, const short wRatio100, const short nThreshold0_01dB, const short nMakeupGain0_01dB)
- ASX32_API [ASX_ERROR](#) [ASX_Compander_Get](#) ([ASX_HANDLE](#) hCompander, unsigned short *pwAttack, unsigned short *pwDecay, short *pwRatio100, short *pnThreshold0_01dB, short *pnMakeupGain0_01dB)
- ASX32_API [ASX_ERROR](#) [ASX_Compander_SetEnable](#) ([ASX_HANDLE](#) hCompander, const unsigned int nOn)

Sets the on/off parameter for the compander.

- ASX32_API [ASX_ERROR](#) [ASX_Compander_GetEnable](#) ([ASX_HANDLE](#) hCompander, unsigned int *nOn)

Gets the on/off parameter for the compander.

- ASX32_API [ASX_ERROR](#) [ASX_Compander_SetMakeupGain](#) ([ASX_HANDLE](#) hCompander, const float fMakeupGain)

Set the compander makeup gain.

- ASX32_API [ASX_ERROR](#) [ASX_Compander_GetMakeupGain](#) ([ASX_HANDLE](#) hCompander, float *fMakeupGain)

Get the compander makeup gain.

- ASX32_API [ASX_ERROR](#) [ASX_Compander_SetAttackTimeConstant](#) ([ASX_HANDLE](#) hCompander, enum [asxCOMPANDER_INDEX](#) index, const unsigned int nAttack)

Set the attack time constant in ms.

- ASX32_API [ASX_ERROR](#) [ASX_Compander_GetAttackTimeConstant](#) ([ASX_HANDLE](#) hCompander, enum [asxCOMPANDER_INDEX](#) index, unsigned int *pnAttack)

Get the attack time constant in ms.

- ASX32_API [ASX_ERROR](#) [ASX_Compander_SetDecayTimeConstant](#) ([ASX_HANDLE](#) hCompander, enum [asxCOMPANDER_INDEX](#) index, const unsigned int nDecay)

Set the decay time constant in ms.

- ASX32_API [ASX_ERROR](#) [ASX_Compander_GetDecayTimeConstant](#) ([ASX_HANDLE](#) hCompander, enum [asxCOMPANDER_INDEX](#) index, unsigned int *pnDecay)
Get the decay time constant in ms.
- ASX32_API [ASX_ERROR](#) [ASX_Compander_SetThreshold](#) ([ASX_HANDLE](#) hCompander, enum [asxCOMPANDER_INDEX](#) index, const float nThreshold)
Set the compander threshold in dbFS.
- ASX32_API [ASX_ERROR](#) [ASX_Compander_GetThreshold](#) ([ASX_HANDLE](#) hCompander, enum [asxCOMPANDER_INDEX](#) index, float *pnThreshold)
Get the compander threshold in dbFS.
- ASX32_API [ASX_ERROR](#) [ASX_Compander_SetRatio](#) ([ASX_HANDLE](#) hCompander, enum [asxCOMPANDER_INDEX](#) index, const float fRatio)
Set the compander ratio (slope).
- ASX32_API [ASX_ERROR](#) [ASX_Compander_GetRatio](#) ([ASX_HANDLE](#) hCompander, enum [asxCOMPANDER_INDEX](#) index, float *fRatio)
Get the compander ratio (slope).
- ASX32_API [ASX_ERROR](#) [ASX_Cobranet_EnumerateModes](#) ([ASX_HANDLE](#) hCobranet, const int nIndex, enum [asxCOBANET_MODE](#) *peMode, int *pnCount)
- ASX32_API [ASX_ERROR](#) [ASX_Cobranet_GetMode](#) ([ASX_HANDLE](#) hCobranet, enum [asxCOBANET_MODE](#) *peMode)
- ASX32_API [ASX_ERROR](#) [ASX_Cobranet_SetMode](#) ([ASX_HANDLE](#) hCobranet, const enum [asxCOBANET_MODE](#) eMode)
- ASX32_API [ASX_ERROR](#) [ASX_Cobranet_GetIPAddress](#) ([ASX_HANDLE](#) hCobranet, unsigned int *pdwIPAddr)
Get the current IP address of the Cobranet device.
- ASX32_API [ASX_ERROR](#) [ASX_Cobranet_SetIPAddress](#) ([ASX_HANDLE](#) hCobranet, const unsigned int dwIPAddr)
Set the current IP address of the Cobranet device.
- ASX32_API [ASX_ERROR](#) [ASX_Cobranet_GetStaticIPAddress](#) ([ASX_HANDLE](#) hCobranet, unsigned int *pdwIPAddr)
Get the static IP address of the Cobranet device.
- ASX32_API [ASX_ERROR](#) [ASX_Cobranet_SetStaticIPAddress](#) ([ASX_HANDLE](#) hCobranet, const unsigned int dwIPAddr)
Set the static IP address of the Cobranet device.
- ASX32_API [ASX_ERROR](#) [ASX_Cobranet_GetMACAddress](#) ([ASX_HANDLE](#) hCobranet, unsigned int *pdwMAC_MSBs, unsigned short *pwMAC_LSBs)
Get the current cobranet MAC address.

- ASX32_API [ASX_ERROR ASX_Cobranet_GetDescription](#) ([ASX_HANDLE](#) hCobranet, char *szString, const int nLength)
Get the device's description from the sysDescr SNMP field.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetName](#) ([ASX_HANDLE](#) hCobranet, char *szString, const int nLength)
Get the device's name from the sysName SNMP field.
- ASX32_API [ASX_ERROR ASX_Cobranet_SetName](#) ([ASX_HANDLE](#) hCobranet, const char *pszLongInputString)
Set the device's name in the sysName SNMP field.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetLocation](#) ([ASX_HANDLE](#) hCobranet, char *szString, const int nLength)
Get the device's location from the sysLocation SNMP field.
- ASX32_API [ASX_ERROR ASX_Cobranet_SetLocation](#) ([ASX_HANDLE](#) hCobranet, const char *pszLongInputString)
Set the device's location in the sysLocation SNMP field.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetFirmwareRevision](#) ([ASX_HANDLE](#) hCobranet, char *pszRevision)
Gets a device's firmware revision.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetErrorInfo](#) ([ASX_HANDLE](#) hCobranet, unsigned int *pnCode, unsigned int *pnCount, unsigned int *pnDisplay)
Gets a device's error information.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetLatencyAndSampleRate](#) ([ASX_HANDLE](#) hCobranet, enum [asxCOBANET_LATENCY](#) *peLatency, enum [asxSAMPLE_RATE](#) *peRate)
Gets a device's latency and sample reate.
- ASX32_API [ASX_ERROR ASX_Cobranet_SetLatencyAndSampleRate](#) ([ASX_HANDLE](#) hCobranet, const enum [asxCOBANET_LATENCY](#) eLatency, const enum [asxSAMPLE_RATE](#) eRate)
Gets a device's latency and sample reate.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetPersistence](#) ([ASX_HANDLE](#) hCobranet, unsigned int *pnSetting)
Gets a device's flash persistence setting.
- ASX32_API [ASX_ERROR ASX_Cobranet_SetPersistence](#) ([ASX_HANDLE](#) hCobranet, const unsigned int nSetting)
Sets a device's flash persistence state.

- ASX32_API [ASX_ERROR ASX_Cobranet_GetConductorPriority](#) (ASX_HANDLE hCobranet, unsigned int *pnPriority)
Gets a device's conductor priority.
- ASX32_API [ASX_ERROR ASX_Cobranet_SetConductorPriority](#) (ASX_HANDLE hCobranet, const unsigned int nPriority)
Sets a device's conductor priority.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetConductorStatus](#) (ASX_HANDLE hCobranet, unsigned int *pnState)
Gets a device's conductor status.
- ASX32_API [ASX_ERROR ASX_Cobranet_SetSerialEnable](#) (ASX_HANDLE hCobranet, const int nOnOff)
Enable or disable a device's serial bridge.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetSerialEnable](#) (ASX_HANDLE hCobranet, int *pOnOff)
Gets a device's serial bridge status.
- ASX32_API [ASX_ERROR ASX_Cobranet_SetSerialConfig](#) (ASX_HANDLE hCobranet, const unsigned int nBaud, const unsigned int nPPeriod, const char pRxMAC[6], const int nAcceptUnicast, const char pTxMAC[6])
Configures a device's serial bridge.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetSerialConfig](#) (ASX_HANDLE hCobranet, unsigned int *pnBaud, unsigned int *pnPPeriod, char pRxMAC[6], int *pnAcceptUnicast, char pTxMAC[6])
Gets a device's serial bridge configuration.
- ASX32_API [ASX_ERROR ASX_Cobranet_GetIfStatus](#) (ASX_HANDLE hCobranet, unsigned int *pnCurrentIf, unsigned int *pnPrimaryLinkStatus, unsigned int *pnSecondaryLinkStatus)
Gets a device's ethernet connection status.
- ASX32_API [ASX_ERROR ASX_CobranetTx_GetStatus](#) (ASX_HANDLE hCobranetTx, unsigned int *pnDropouts, unsigned int *pnPosition, unsigned int *pnReceivers)
Gets a Cobranet transmitter's status.
- ASX32_API [ASX_ERROR ASX_CobranetTx_GetBundle](#) (ASX_HANDLE hCobranetTx, unsigned int *pnBundle)
Gets a Cobranet transmitter's bundle.
- ASX32_API [ASX_ERROR ASX_CobranetTx_SetBundle](#) (ASX_HANDLE hCobranetTx, const unsigned int nBundle)

Sets a Cobranet transmitter's bundle.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetTx_GetChannelCount](#) ([ASX_HANDLE](#) hCobranetTx, unsigned int *pnCount)

Gets a Cobranet transmitter's channel count.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetTx_SetChannelCount](#) ([ASX_HANDLE](#) hCobranetTx, const unsigned int nCount)

Sets a Cobranet transmitter's channel count.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetTx_GetChannelMap](#) ([ASX_HANDLE](#) hCobranetTx, unsigned int nMap[8])

Gets a Cobranet transmitter's channel map.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetTx_SetChannelMap](#) ([ASX_HANDLE](#) hCobranetTx, const unsigned int nMap[8])

Sets a Cobranet transmitter's channel map.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetTx_GetFormat](#) ([ASX_HANDLE](#) hCobranetTx, enum [asxAUDIO_FORMAT](#) *peFormat)

Gets a Cobranet transmitter's sub format map.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetTx_SetFormat](#) ([ASX_HANDLE](#) hCobranetTx, const enum [asxAUDIO_FORMAT](#) eFormat)

Sets a Cobranet transmitter's channel format.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetTx_GetUnicastMode](#) ([ASX_HANDLE](#) hCobranetTx, unsigned int *pnUnicastMode, unsigned int *pnMaxUnicast)

Gets a Cobranet transmitter's unicast information.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetTx_SetUnicastMode](#) ([ASX_HANDLE](#) hCobranetTx, const unsigned int nUnicastMode, const unsigned int nMaxUnicast)

Sets a Cobranet transmitter's unicast information.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetRx_GetStatus](#) ([ASX_HANDLE](#) hCobranetRx, unsigned int *pnStatus, unsigned int *pnDropouts, unsigned int *pnDelay, unsigned int nFormat[8])

Gets a Cobranet receiver's status.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetRx_GetBundle](#) ([ASX_HANDLE](#) hCobranetRx, unsigned int *pnBundle)

Gets a Cobranet receiver's bundle.

- ASX32_API [ASX_ERROR](#) [ASX_CobranetRx_SetBundle](#) ([ASX_HANDLE](#) hCobranetRx, const unsigned int nBundle)

Sets a Cobranet receiver's bundle.

- ASX32_API [ASX_ERROR ASX_CobranetRx_GetSourceMAC](#) (ASX_HANDLE hCobranetRx, unsigned int *pdwMAC_MSBs, unsigned short *pwMAC_LSBs)

Gets a Cobranet receiver's source MAC address for private bundles.

- ASX32_API [ASX_ERROR ASX_CobranetRx_SetSourceMAC](#) (ASX_HANDLE hCobranetRx, const unsigned int dwMAC_MSBs, const unsigned short wMAC_LSBs)

Sets a Cobranet receiver's source MAC address for private bundles.

- ASX32_API [ASX_ERROR ASX_CobranetRx_GetChannelMap](#) (ASX_HANDLE hCobranetRx, unsigned int nMap[8])

Gets a Cobranet receiver's channel mapping.

- ASX32_API [ASX_ERROR ASX_CobranetRx_SetChannelMap](#) (ASX_HANDLE hCobranetRx, const unsigned int nMap[8])

Sets a Cobranet receiver's channel mapping.

- ASX32_API [ASX_ERROR ASX_CobranetRx_GetMinimumDelay](#) (ASX_HANDLE hCobranetRx, unsigned int *pnMinDelay)

Gets a Cobranet receiver's minimum delay.

- ASX32_API [ASX_ERROR ASX_CobranetRx_SetMinimumDelay](#) (ASX_HANDLE hCobranetRx, const unsigned int nMinDelay)

Sets a Cobranet receiver's channel mapping.

- ASX32_API [ASX_ERROR ASX_ToneDetector_SetEnable](#) (ASX_HANDLE hToneDetector, const unsigned int nEnable)

Turns the entire tone detector on and off.

- ASX32_API [ASX_ERROR ASX_ToneDetector_GetEnable](#) (ASX_HANDLE hToneDetector, unsigned int *nEnable)

Returns whether the entire tone detector is on or off.

- ASX32_API [ASX_ERROR ASX_ToneDetector_SetEventEnable](#) (ASX_HANDLE hToneDetector, const unsigned int nEnable)

Turns the event reporting function of the tone detector on and off.

- ASX32_API [ASX_ERROR ASX_ToneDetector_GetEventEnable](#) (ASX_HANDLE hToneDetector, unsigned int *nEnable)

Returns whether the event reporting function of the tone detector is on or off.

- ASX32_API [ASX_ERROR ASX_ToneDetector_SetThreshold](#) (ASX_HANDLE hToneDetector, const float fThreshold)

Sets the tone detector threshold (units of dB)

- ASX32_API [ASX_ERROR](#) [ASX_ToneDetector_GetThreshold](#) (ASX_HANDLE hToneDetector, float *fThreshold)
Gets the tone detector threshold (units of dB) with respect to full scale eg.
- ASX32_API [ASX_ERROR](#) [ASX_ToneDetector_GetState](#) (ASX_HANDLE hToneDetector, unsigned int *nState)
Gets the tone detector state.
- ASX32_API [ASX_ERROR](#) [ASX_ToneDetector_GetFrequency](#) (ASX_HANDLE hToneDetector, unsigned int nIndex, unsigned int *nState)
Gets the centre frequency of each tone detector by index.
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_SetEnable](#) (ASX_HANDLE hSilenceDetector, const unsigned int nEnable)
Turns the entire silence detector on and off.
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_GetEnable](#) (ASX_HANDLE hSilenceDetector, unsigned int *nEnable)
Returns whether the entire silence detector is on or off.
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_SetEventEnable](#) (ASX_HANDLE hSilenceDetector, const unsigned int nEnable)
Turns the event reporting function of the silence detector on and off.
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_GetEventEnable](#) (ASX_HANDLE hSilenceDetector, unsigned int *nEnable)
Returns whether the event reporting function of the silence detector is on or off.
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_SetDelay](#) (ASX_HANDLE hSilenceDetector, const unsigned int Delay)
Set the silence detector delay.
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_GetDelay](#) (ASX_HANDLE hSilenceDetector, unsigned int *Delay)
Get the silence detector delay.
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_SetThreshold](#) (ASX_HANDLE hSilenceDetector, const float fThreshold)
Sets the silence detector threshold (units of dB)
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_GetThreshold](#) (ASX_HANDLE hSilenceDetector, float *fThreshold)
Gets the silence detector threshold (units of dB)
- ASX32_API [ASX_ERROR](#) [ASX_SilenceDetector_GetState](#) (ASX_HANDLE hSilenceDetector, unsigned int *nState)
Gets the silence detector state.

- ASX32_API [ASX_ERROR](#) [ASX_Block_GetInfo](#) ([ASX_HANDLE](#) hBlock, char *szBlockName, const unsigned int uStringLength, unsigned int *uParameterCount)

Gets the name of the block and the number of parameters it has.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_GetName](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, char *szParameterName, const unsigned int uStringLength)

Gets the name of parameter number uParameterIndex.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_GetUnits](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, char *szParameterUnits, const unsigned int uStringLength)

Gets the units of parameter number uParameterIndex.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_GetType](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, enum [asxUCONTROL_PTYPE](#) *eType)

Gets the type of parameter number uParameterIndex.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_GetFlags](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, enum [asxUCONTROL_PFLAGS](#) *eFlags)

Gets the flags for parameter number uParameterIndex.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_GetElementCount](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, unsigned int *uCount)

Gets the number of elements.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_GetRange](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, struct [asxParameterRangeInfo](#) *info)

Gets the parameter range.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_GetEnumName](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, const unsigned int uEnumItem, char *szEnumName, const unsigned int uStringLength)

Gets the enumerated names for a parameter.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_Set](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, struct [asxParameterValue](#) *data)

Sets a parameter's value field.

- ASX32_API [ASX_ERROR](#) [ASX_Block_Parameter_Get](#) ([ASX_HANDLE](#) hBlock, const unsigned int uParameterIndex, struct [asxParameterValue](#) *data)

Gets a parameter's value field.

9.1.1 Define Documentation

9.1.1.1 `#define _RPT0( l, s ) printf(s)`

A debug helper function, 0 arguments.

9.1.1.2 `#define _RPT1( l, s, d1 ) printf(s,d1)`

A debug helper function, 1 argument.

9.1.1.3 `#define ARRAY_SIZE( X ) (sizeof(X)/sizeof(X[0]))`

9.1.1.4 `#define ASX32_API`

9.1.1.5 `#define ASX_LONG_STRING 128`

Long string size for error strings, filenames and PADs strings.

Examples:

[cobranet/main.c](#).

9.1.1.6 `#define ASX_LONGLONG_STRING 256`

LongLong string size for PADs comment string.

Examples:

[tuner/main.c](#).

9.1.1.7 `#define ASX_SHORT_STRING 32`

Short string size for adapter, node, control, enum translations.

Examples:

[adapter/main.c](#), [cobranet/main.c](#), and [volume/main.c](#).

9.1.2 Typedef Documentation

9.1.2.1 `typedef enum asxERROR ASX_ERROR`

Error type used to return error codes from all functions.

9.1.2.2 `typedef void ASX_ERROR_CALLBACK(ASX_HANDLE hASX_Object, const char *pszCallingFunction, void *pUser1, void *pUser2)`

An error handling callback function.

9.1.2.3 `typedef void* ASX_HANDLE`

Generic handle used to represent all ASX objects.

9.1.2.4 `typedef enum asxNODE ASX_NODE`

Node type enum.

9.1.2.5 `typedef void ASX_PLAYER_CALLBACK(ASX_HANDLE hASX_Player_Object, const enum asxPLAYER_FLAGS flags, void *pUser1)`

A playback callback function.

9.1.2.6 `typedef int ASX_TIME`

Timescale.

9.1.3 Enumeration Type Documentation

9.1.3.1 `enum asxADAPTER_PROPERTY`

Properties for use with ASX_Adapter_ReadProperty and ASX_Adapter_WriteProperty.

Enumerator:

asxADAPTER_PROPERTY_ERRATA_1 true if errata_1 workaround for 6100 cards is turned on.

asxADAPTER_PROPERTY_SX2_SETTING true when SX2 is on

asxADAPTER_PROPERTY_SYNC_HEADER_CONNECTIONS (read-only), the number of headers connected.

asxADAPTER_PROPERTY_SUPPORT_SX2 (read-only), returns true or false.

asxADAPTER_PROPERTY_SUPPORTS_FW_UPDATE (read-only), device supports firmware updating

asxADAPTER_PROPERTY_FIRMWARE_ID (read-only), firmware ID

asxADAPTER_PROPERTY_SUPPORTS_SNMP (read-only), device supports SNMP

9.1.3.2 enum asxADAPTERMODE

Adapter mode settings.

Enumerator:

- asxADAPTERMODE_ILLEGAL* Illegal adapter mode.
- asxADAPTERMODE_4_PLAY* Adapter has 4 playback streams.
- asxADAPTERMODE_6_PLAY* Adapter has 6 playback streams.
- asxADAPTERMODE_8_PLAY* Adapter has 8 playback streams.
- asxADAPTERMODE_9_PLAY* Adapter has 9 playback streams.
- asxADAPTERMODE_12_PLAY* Adapter has 12 playback streams.
- asxADAPTERMODE_16_PLAY* Adapter has 16 playback streams.
- asxADAPTERMODE_1_PLAY* Adapter has 1 playback streams.
- asxADAPTERMODE_MODE_1* Adapter mode 1. Exact meaning depends on the adapter.
- asxADAPTERMODE_MODE_2* Adapter mode 2. Exact meaning depends on the adapter.
- asxADAPTERMODE_MODE_3* Adapter mode 3. Exact meaning depends on the adapter.
- asxADAPTERMODE_MULTICHANNEL* Adapter set to handle streams with more than 2 channels.
- asxADAPTERMODE_MONO* Adapter set to handle mono streams, including physical I/O.
- asxADAPTERMODE_LOW_LATENCY* Adapter set to low latency mode.
- asxADAPTERMODE_24_PLAY* Adapter has 24 playback streams.
- asxADAPTERMODE_32_PLAY* Adapter has 32 playback streams.

9.1.3.3 enum asxADPROPENUM_MODE

Adapter property enumerate mode settings.

Enumerator:

- asxADPROPENUM_MODE_PROPERTIES* Enumerate adapter properties.
- asxADPROPENUM_MODE_SETTINGS* Enumerate adapter property settings.

9.1.3.4 enum asxADPROPENUM_SSX2

Adapter property SSX2 enumerate settings.

Enumerator:

- asxADPROPENUM_SSX2_OFF* SSX2 off.
- asxADPROPENUM_SSX2_ON* SSX2 on.

9.1.3.5 enum asxAESEBU_FORMAT

Digital mode settings.

Enumerator:

- asxAESEBU_FORMAT_AESEBU* AES/EBU format is set to AES/EBU (professional)
- asxAESEBU_FORMAT_SPDIF* AES/EBU format is set to S/PDIF (consumer)
- asxAESEBU_FORMAT_UNDEFINED* AES/EBU format is undefined.

9.1.3.6 enum asxAESEBU_STATUS

AESEBU status bitfields. Not translatable to strings.

Enumerator:

- asxAESEBU_ERROR* AESEBU error.
- asxAESEBU_ERROR_NOT_LOCKED* AESEBU not locked to input signal.
- asxAESEBU_ERROR_POOR_QUALITY* AESEBU signal is poor quality.
- asxAESEBU_ERROR_PARITY_ERROR* AESEBU parity error was detected.
- asxAESEBU_ERROR_BIPHASE_VIOLATION* AESEBU Biphase violation.
- asxAESEBU_ERROR_VALIDITY* AESEBU validity error.
- asxAESEBU_ERROR_CHANNELSTATUS_CRC* AESEBU channel status error.

9.1.3.7 enum asxAUDIO_FORMAT

Audio Formats.

Enumerator:

- asxAUDIO_FORMAT_PCM8* 8-bit PCM, unsigned.
- asxAUDIO_FORMAT_PCM16* 16-bit PCM, signed.
- asxAUDIO_FORMAT_PCM24* 24-bit PCM, signed.
- asxAUDIO_FORMAT_PCM32* 32-bit PCM, signed.
- asxAUDIO_FORMAT_PCM32_FLOAT* 32-bit PCM in IEEE float format.
- asxAUDIO_FORMAT_MPEG_L2* MPEG-1, Layer-II.
- asxAUDIO_FORMAT_MPEG_L3* MPEG-1, Layer-III, or "mp3".
- asxAUDIO_FORMAT_MPEG_AACPLUS* AAC+.
- asxAUDIO_FORMAT_DOLBY_AC2* Dolby AC-2.
- asxAUDIO_FORMAT_PCM20* 20-bit PCM, signed.
- asxAUDIO_FORMAT_NONE* Unspecified or invalid audio format.

9.1.3.8 enum asxCHANNELMODE

Channel mode settings.

Enumerator:

- asxCHANNELMODE_ILLEGAL* Illegal channel mode.
- asxCHANNELMODE_NORMAL* Normal mode. Left goes to left and right goes to right.
- asxCHANNELMODE_SWAP* Channels are swapped. Left goes to right and right goes to left.
- asxCHANNELMODE_STEREOTOLEFT* Both left and right channels are merged and output on left channel.
- asxCHANNELMODE_STEREOTORIGHT* Both left and right channels are merged and output on the right channel.
- asxCHANNELMODE_LEFTTOSTEREO* The left channel is converted to a stereo output. Right input channel is discarded.
- asxCHANNELMODE_RIGHTTOSTEREO* The right channel is converted to a stereo output. Left input channel is discarded.

9.1.3.9 enum asxCOBANET_IFSTATUS

Cobranet If status bitfields. Not translatable to strings.

Enumerator:

- asxCOBANET_IFSTATUS_LINK_ESTABLISHED* Ethernet link established.
- asxCOBANET_IFSTATUS_FULL_DUPLEX* Connection is full-duplex.
- asxCOBANET_IFSTATUS_ACTIVE_CONNECTION* Packets being received at least every other second.

9.1.3.10 enum asxCOBANET_LATENCY

Cobranet latency settings.

Enumerator:

- asxCOBANET_LATENCY_133ms* Cobranet latency of 1 1/3 ms.
- asxCOBANET_LATENCY_266ms* Cobranet latency of 2 2/3 ms.
- asxCOBANET_LATENCY_533ms* Cobranet latency of 5 1/3 ms.

9.1.3.11 enum asxCOBNET_MODE

Cobranet mode settings (deprecated!)

Enumerator:

asxCOBNET_MODE_NETWORK Cobranet mode is set to networked.

asxCOBNET_MODE_TETHERED Cobranet mode is set to tethered.

9.1.3.12 enum asxCOMPANDER_INDEX

Compander control indices.

Enumerator:

asxCOMPANDER_INDEX_NOISEGATE Noise gate index.

asxCOMPANDER_INDEX_COMPANDER Compander index.

9.1.3.13 enum asxCONTROL

Control type identifiers. The control types are used to differentiate control capabilities.

Enumerator:

asxCONTROL_INVALID Placeholder that indicates an error.

asxCONTROL_CONNECTION Documents a connection, no functionality.

asxCONTROL_VOLUME A volume control. A large number of volume controls are used to control audio routing and mixing.

asxCONTROL_METER A meter control. This is used to implement peak meter functionality.

asxCONTROL_MUTE A mute control.

asxCONTROL_MULTIPLEXER A multiplexer control allows the selection of a single audio source from many possible sources.

asxCONTROL_AESEBU_TRANSMITTER An AESEBU transmitter control.

asxCONTROL_AESEBU_RECEIVER An AESEBU receiver control.

asxCONTROL_LEVEL A level/trim control for adjusting input and output levels.

asxCONTROL_TUNER A tuner control.

asxCONTROL_RDS An FM RDS/RBDS control.

asxCONTROL_VOX A Vox control used to set record trigger level.

asxCONTROL_AES18_TRANSMITTER An AES18 transmitter control.

asxCONTROL_AES18_RECEIVER An AES18 receiver control.

asxCONTROL_AES18_BLOCK_GENERATOR An AES18 block generator control.

asxCONTROL_CHANNEL_MODE A channel mode control for adjusting audio routing.

asxCONTROL_BIT_STREAM A bitstream control for setting raw bitstream clocking parameters.

asxCONTROL_SAMPLE_CLOCK A sample clock control.

asxCONTROL_MICROPHONE A microphone control.

asxCONTROL_PARAMETRIC_EQ A parametric EQ control.

asxCONTROL_COMPANDER A compander control.

asxCONTROL_COBRANET A cobranet control.

asxCONTROL_PLAYER A player control that can play audio files from disk.

asxCONTROL_RECORDER A recorder control that can record audio files to disk.

asxCONTROL_GPIO A general purpose input/output control that typically would manipulate relays and read opto inputs.

asxCONTROL_RESERVED_525 Reserved for future use.

asxCONTROL_RESERVED_526 Reserved for future use.

asxCONTROL_RESERVED_527 Reserved for future use.

asxCONTROL_RESERVED_528 Reserved for future use.

asxCONTROL_GENERIC A generic control used as a placeholder during development.

asxCONTROL_TONEDETECTOR A tone detector control.

asxCONTROL_SILENCEDETECTOR A Silence detector control.

asxCONTROL_COBRANET_TRANSMITTER A Cobranet transmitter control.

asxCONTROL_COBRANET_RECEIVER A Cobranet receiver control.

asxCONTROL_PAD A Program Auxiliary Data including RBDS control.

asxCONTROL_SRC Samplerate converter control.

asxCONTROL_BLOCK Block control.

asxCONTROL_LAST_CONTROL

9.1.3.14 enum **asxEQBANDTYPE**

Parametric equalizer band type settings.

Enumerator:

asxEQBANDTYPE_BYPASS Bypass.

asxEQBANDTYPE_LOWSHELF Low Shelf - programmed gain below freq, unity gain above.

asxEQBANDTYPE_HIGHSHELF High Shelf - programmed gain above freq, unity gain below.

asxEQBANDTYPE_EQUALIZER Equalizer - programmed gain in passband, unity gain outside passband.

asxEQBANDTYPE_LOWPASS Low Pass - unity gain below freq, attenuated above.

asxEQBANDTYPE_HIGHPASS High Pass - unity gain above freq, attenuated below.

asxEQBANDTYPE_BANDPASS Band Pass - unity gain in passband, attenuated outside passband.

asxEQBANDTYPE_BANDSTOP Band Stop - attenuated in passband, unity gain outside passband.

9.1.3.15 enum asxERROR

ASX error codes. These error codes are returned by most ASX functions.

Enumerator:

asxERROR_NO_ERROR The success, or no error code is 0.

asxERROR_ASXObject An attempt was made to call an ASX function with an incorrect object handle.

asxERROR_INDEX_OUT_OF_RANGE The index passed in to the function is out of range.

asxERROR_UNIMPLEMENTED An attempt was made to call an un-implemented function.

asxERROR_COMMUNICATING_WITH_DEVICE Device communication error.

asxERROR_STARTING_DEVICE Device would not start. Typically this is a driver installation or hardware problem.

asxERROR_NOT_OPEN An attempt was made to manipulate an ASX object that requires opening before use.

asxERROR_ALREADY_OPEN An attempt was made to open an object that is already open.

asxERROR_INVALID_FORMAT The format is illegal on this adapter. It may be the sample rate or the compression format that is illegal.

asxERROR_INTERNAL_BUFFERING_ERROR Buffering error internal to the ASX library occurred.

asxERROR_AES18 AES-18 signalling error.

asxERROR_INVALID_OPERATION An attempt was made to perform an invalid operation.

asxERROR_ENUMERATE_INDEX_OUT_OF_RANGE The index passed in to the enumeration function is too large.

asxERROR_BUFFER_TOO_SMALL A buffer was passed to a function that was too small to hold the requested data.

asxERROR_OUTOFMEMORY An internal system call to allocate memory failed.

asxERROR_DEPRECATED An attempt was made to call a deprecated function.

asxERROR_TOO_MANY_CLIENTS Too many network clients communicating with the device.

asxERROR_COBRANET_NODE_NOT_FOUND AsiCnDisco has not found this node.

asxERROR_COBRANET_NODE_FOUND AsiCnDisco has found this node but hasn't assigned an IP address.

asxERROR_NO_IP_ADDRESSES_AVAILABLE The entire range specified for IP autoassignment is in use.

asxERROR_IP_ASSIGNED AsiCnDisco tried to assign an address to this CobraNet node.

asxERROR_IP_CHANGED The IP address for this CobraNet node has changed, it should be non-zero.

asxERROR_IP_AUTOASSIGN_DISABLED The IP address auto-assign feature is disabled.

asxERROR_PCAP_ERROR The pcap driver failed to start.

asxERROR_DISCO_DLL_NOT_FOUND The asicndiscoXX.dll failed to load.

asxERROR_HOST_NOT_FOUND The specified host IP address is not in the list of available adapters.

asxERROR_COBRANET_NODE_UNREACHABLE The node is on another subnet and is unreachable via SNMP.

asxERROR_DUPLICATE_ADAPTER_INDEX The HPI adapter has a duplicate index.

asxERROR_INVALID_CONTROL An attempt was made to use an invalid control.

asxERROR_INVALID_CONTROL_VALUE An attempt was made to set a control to an invalid value.

asxERROR_INVALID_CONTROL_NOT_FOUND The specified control could not be found.

asxERROR_INVALID_NUMBER_OF_CHANNELS An invalid number of channels were passed in.

asxERROR_INVALID_CONTROL_ATTRIBUTE An attempt was made to set an attribute (or property) of a control that does not exist.

asxERROR_UNSUPPORTED_CONTROL_ATTRIBUTE Control attribute or function is not supported by this hardware.

asxERROR_INVALID_CONTROL_OPERATION Control does not support the requested operation.

asxERROR_CONTROL_NOT_READY Control is not ready for the requested operation.

asxERROR_FILE_OPEN_FAILED File open failed.

asxERROR_PLAYER_INTERNAL_STATE_FAILURE Player control had an internal state error - contact AudioScience.

asxERROR_PLAYER_TIME_OUT Player control had an internal timeout error - contact AudioScience.

asxERROR_PLAYER_OUT_OF_SEQUENCE_CALL An attempt was made to perform an operation that is illegal in the current player state.

asxERROR_PLAYER_TWAV Internal player error.

asxERROR_PLAYER_NOFILE Operation requires an open filehandle and no file is open.

asxERROR_PLAYER_INVALIDFILEFORMAT File is invalid or corrupt.

asxERROR_PLAYER_UNSUPPORTEDFORMAT File is in an unsupported format.

asxERROR_PLAYER_FILEREADERROR An error occurred when reading the file.

asxERROR_PLAYER_FILEOPENERROR Could not open the file. It may be missing.

asxERROR_RECORDER_INTERNAL_STATE_FAILURE Recorder control had an internal state error - contact AudioScience.

asxERROR_RECORDER_TIME_OUT Recorder control had an internal timeout error - contact AudioScience.

asxERROR_RECORDER_OUT_OF_SEQUENCE_CALL An attempt was made to perform an operation that is illegal in the current recorder state.

asxERROR_RECORDER_TWAV Internal recorder error.

asxERROR_RECORDER_FILECREATEERROR Couldn't create file, could be write protected.

asxERROR_RECORDER_FILEWRITEERROR An error occurred when writing the file.

asxERROR_RECORDER_FORMATMISMATCH Tried to append to a file with a different format.

asxERROR_RECORDER_INVALIDFILENAME Tried to record a WAV file without the .WAV extension or tried to record a RAW file with the .WAV extension.

asxERROR_MIXER_SAVECONTROLSTATE Mixer command to save controls on device failed.

asxERROR_UNKNOWN Unknown error.

9.1.3.16 enum `asxFILE_FORMAT`

File Formats.

Enumerator:

`asxFILE_FORMAT_WAV` Standard Windows .WAV soundfile.

`asxFILE_FORMAT_RAW` Raw binary data (no format header).

9.1.3.17 enum `asxFILE_MODE`

File Mode.

Enumerator:

`asxFILE_MODE_CREATE` File is created, if it exists it is overwritten.

`asxFILE_MODE_APPEND` File is appended if it exists, created if it doesn't.

9.1.3.18 enum `asxHANDLE_TYPE`

Handle type enums returned from [ASX_Handle_GetType\(\)](#).

Enumerator:

`asxHANDLE_INVALID` Invalid handle.

`asxHANDLE_SYSTEM` Handle to ASX system.

`asxHANDLE_ADAPTER` Handle to ASX adapter.

`asxHANDLE_MIXER` Handle to ASX mixer.

`asxHANDLE_NODE` Handle to ASX node.

`asxHANDLE_CONTROL` Handle to ASX control.

`asxHANDLE_LAST` Last handle placeholder.

9.1.3.19 enum `asxMETER_TYPE`

Peak meter type to read.

Enumerator:

`asxMETER_PEAK` The peak level of the signal.

`asxMETER_RMS` The RMS level of the signal.

9.1.3.20 enum asxMSG_LOGGING

Error logging control. Uses DbgView under Windows to log messages.

Enumerator:

- asxMSG_LOGGING_DISABLE* Disable all message logging.
- asxMSG_LOGGING_ERROR* Enable logging of error messages.
- asxMSG_LOGGING_WARNING* Enable logging of warning and error messages.
- asxMSG_LOGGING_NOTICE* Enable logging of notice (ASX function error returns), warning and error messages.
- asxMSG_LOGGING_INFO* Enable logging of info (ASX function calls), notice, warning and error messages.
- asxMSG_LOGGING_DEBUG* Enable logging of debug, info, notice, warning and error messages.
- asxMSG_LOGGING_VERBOSE* Enable logging of all messages.

9.1.3.21 enum asxNODE

Node type identifiers. The nodes identify how controls are connected and located. This enum is used to identify node types.

Enumerator:

- asxNODE_NONE* Node does not exist value.
- asxNODE_INVALID* Node is invalid.
- asxNODE_ADAPTER* An adapter node that contains controls like GPIO and NvMem.
- asxNODE_PLAYER* A player source node. This node has the ability to play audio files.
- asxNODE_LINE_IN* An audio source coming from a line in audio connector.
- asxNODE_AESEBU_IN* An audio source coming from a digital input. Typically this will be an AESEBU/SPDIF connection.
- asxNODE_TUNER_IN* An audio source coming from a tuner.
- asxNODE_RADIO_FREQ_IN* Not implemented.
- asxNODE_CLOCK_SOURCE_IN* A source for of adapter sample clock generation.
- asxNODE_BITSTREAM_IN* A raw, unformatted bitstream. To date this has been used to represent an RS422 serial stream fed directly from a satellite receiver.
- asxNODE_MICROPHONE_IN* An audio source coming from a microphone.
- asxNODE_COBRANET_IN* An audio source coming from a Cobranet audio routing channel.

asxNODE_COBRANET_RECEIVER An Cobranet receiver.
asxNODE_ANALOG_IN An Analog input.
asxNODE_SDI_IN An 3G/HD/SD-SDI input.
asxNODE_RTP_DESTINATION_IN An RTP stream destination.
asxNODE_INTERNAL_IN An audio node internal to the device.
asxNODE_AVB_IN_STREAM An AVB input stream.
asxNODE_BLULINK_IN A BLU link input node.
asxNODE_AVB_IN_AUDIO AVB input audio.
asxNODE_LAST_SOURCE_NODE A placekeeper marking the last source node.

asxNODE_FIRST_DEST_NODE A placekeeper marking the last source node.

asxNODE_RECORDER A destination that indicates a path for recording audio to the host.
asxNODE_LINE_OUT A destination that outputs to a line out audio connector.

asxNODE_AESEBU_OUT A destination that outputs to a digital out audio connector. Typically this will be an AESEBU/SPDIF connection.
asxNODE_RADIO_FREQ_OUT Not implemented.
asxNODE_SPEAKER_OUT A destination that outputs to a speaker.
asxNODE_COBRANET_OUT A destination Cobranet audio routing channel.
asxNODE_COBRANET_TRANSMITTER A Cobranet transmitter.
asxNODE_ANALOG_OUT An Analog output.
asxNODE_SDI_OUT An 3G/HD/SD-SDI output.
asxNODE_RTP_SOURCE_OUT An RTP stream source.
asxNODE_AVB_OUT_STREAM An AVB output stream.
asxNODE_INTERNAL_OUT An internal output node.
asxNODE_BLULINK_OUT An BLU link output.
asxNODE_AVB_OUT_AUDIO AVB output audio.
asxNODE_LAST_DEST_NODE A placekeeper marking the last destination node.

9.1.3.22 enum *asxPLAYER_FLAGS*

Player callback flags that form a bitmask. ie they are numbered 1,2,4,8 etc.

Enumerator:

asxPLAYER_FILE_COMPLETE A single playback file has completed.
asxPLAYER_FILELIST_COMPLETE A playlist has completed.
asxPLAYER_FILE_START A single playback file has started.

9.1.3.23 enum asxPLAYER_STATE

Player States.

Enumerator:

- asxPLAYER_INIT* Initialized state.
- asxPLAYER_OPEN* Player is open.
- asxPLAYER_PREFILL* Buffer prefill state.
- asxPLAYER_RUNNING* Player is running, i.e. audio is playing.
- asxPLAYER_PAUSED* Player is paused.
- asxPLAYER_DONE* Player has completed playing a file.
- asxPLAYER_DESTROY* Player is being destroyed.

9.1.3.24 enum asxRECORD_MODE

Record Mode.

Enumerator:

- asxRECORD_MODE_STEREO* MPEG-1 Layer-2 mode stereo.
- asxRECORD_MODE_JOINT_STEREO* MPEG-1 Layer-2 mode joint stereo.
This allows channels to be merged at some frequencies.
- asxRECORD_MODE_DUAL_MONO* MPEG-1 Layer-2 mode dual mono. Maintain left and right channel separation. Requires a higher bitrate.
- asxRECORD_MODE_MONO* MPEG-1 Layer-2 mode mono.
- asxRECORD_MODE_DONT_CARE* Use this mode for PCM files and formats that don't have a "mode".

9.1.3.25 enum asxRECORDER_STATE

Recorder States.

Enumerator:

- asxRECORDER_INIT* Initialized state.
- asxRECORDER_OPEN* Recorder is open.
- asxRECORDER_RUNNING* Recorder is recording to a file.
- asxRECORDER_PAUSED* Recorder is paused.
- asxRECORDER_DONE* Recorder is halted and is no longer recording.
- asxRECORDER_DESTROY* Recorder is being destroyed.

9.1.3.26 enum asxSAMPLE_CLOCK_SOURCE

Sample clock source options.

Enumerator:

- asxSAMPLE_CLOCK_SOURCE_ADAPTER* /deprecated Use asxSAMPLE_CLOCK_SOURCE_LOCAL
- asxSAMPLE_CLOCK_SOURCE_AESEBUSYNC* Sample clock source is derived from an AESEBU sync input.
- asxSAMPLE_CLOCK_SOURCE_WORD* Sample clock source is derived from external word clock connector.
- asxSAMPLE_CLOCK_SOURCE_WORD_HEADER* Sample clock source is derived from word clock header on the adapter.
- asxSAMPLE_CLOCK_SOURCE_SMPTE* Sample clock source is derived from SMPTE.
- asxSAMPLE_CLOCK_SOURCE_NETWORK* Sample clock source is derived from the network.
- asxSAMPLE_CLOCK_SOURCE_AESEBUAUTO* Sample clock source is derived from the first AESEBU input with valid input.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT1* Sample clock source is derived from AESEBU input 1.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT2* Sample clock source is derived from AESEBU input 2.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT3* Sample clock source is derived from AESEBU input 3.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT4* Sample clock source is derived from AESEBU input 4.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT5* Sample clock source is derived from AESEBU input 5.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT6* Sample clock source is derived from AESEBU input 6.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT7* Sample clock source is derived from AESEBU input 7.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT8* Sample clock source is derived from AESEBU input 8.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT9* Sample clock source is derived from AESEBU input 9.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT10* Sample clock source is derived from AESEBU input 10.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT11* Sample clock source is derived from AESEBU input 11.
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT12* Sample clock source is derived from AESEBU input 12.

<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT13</i>	Sample clock source is derived from AESEBU input 13.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT14</i>	Sample clock source is derived from AESEBU input 14.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT15</i>	Sample clock source is derived from AESEBU input 15.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT16</i>	Sample clock source is derived from AESEBU input 16.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT17</i>	Sample clock source is derived from AESEBU input 17.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT18</i>	Sample clock source is derived from AESEBU input 18.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT19</i>	Sample clock source is derived from AESEBU input 19.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT20</i>	Sample clock source is derived from AESEBU input 20.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT21</i>	Sample clock source is derived from AESEBU input 21.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT22</i>	Sample clock source is derived from AESEBU input 22.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT23</i>	Sample clock source is derived from AESEBU input 23.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT24</i>	Sample clock source is derived from AESEBU input 24.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT25</i>	Sample clock source is derived from AESEBU input 25.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT26</i>	Sample clock source is derived from AESEBU input 26.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT27</i>	Sample clock source is derived from AESEBU input 27.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT28</i>	Sample clock source is derived from AESEBU input 28.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT29</i>	Sample clock source is derived from AESEBU input 29.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT30</i>	Sample clock source is derived from AESEBU input 30.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT31</i>	Sample clock source is derived from AESEBU input 31.
<i>asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT32</i>	Sample clock source is derived from AESEBU input 32.
<i>asxSAMPLE_CLOCK_SOURCE_LOCAL</i>	Sample clock source is local PLL.
<i>asxSAMPLE_CLOCK_SOURCE_PREV_MODULE</i>	Sample clock source is previous module daisy-chain.

asxSAMPLE_CLOCK_SOURCE_UNDEFINED Sample clock source is undefined.

asxSAMPLE_CLOCK_SOURCE_LIVEWIRE /deprecated Use *asxSAMPLE_CLOCK_SOURCE_NETWORK*.

asxSAMPLE_CLOCK_SOURCE_BLULINK Sample clock source is generated by blu link.

9.1.3.27 enum *asxSAMPLE_RATE*

Sample rate options.

Enumerator:

asxSAMPLE_RATE_8000
asxSAMPLE_RATE_11025
asxSAMPLE_RATE_16000
asxSAMPLE_RATE_22050
asxSAMPLE_RATE_24000
asxSAMPLE_RATE_32000
asxSAMPLE_RATE_44100
asxSAMPLE_RATE_48000
asxSAMPLE_RATE_64000
asxSAMPLE_RATE_88200
asxSAMPLE_RATE_96000
asxSAMPLE_RATE_192000
asxSAMPLE_RATE_12000
asxSAMPLE_RATE_176400
asxSAMPLE_RATE_UNDEFINED

9.1.3.28 enum *asxTIMESCALE*

TimeScale type identifiers.

Enumerator:

asxTIMESCALE_INVALID Placeholder that indicates an error.
asxTIMESCALE_BYTES Time scale is represented in bytes.
asxTIMESCALE_MILLISECONDS Time scale is represented in milli-seconds.

asxTIMESCALE_SAMPLES Time scale is represented in samples.
asxTIMESCALE_BYTES_REMAINING Time scale is represented in bytes remaining.

asxTIMESCALE_MILLISECONDS_REMAINING Time scale is represented in milli-seconds remaining.

asxTIMESCALE_SAMPLES_REMAINING Time scale is represented in samples remaining.

9.1.3.29 enum asxTUNER_RDS_TYPE

Tuner PSD/PAD/RDS/RBDS type. Not translatable to strings.

Enumerator:

asxTUNER_RDS_TYPE_RDS RDS data.

asxTUNER_RDS_TYPE_RBDS RBDS data (USA) and HD Radio PSD.

9.1.3.30 enum asxTUNER_STATUS

Tuner status bitfields. Not translatable to strings.

Enumerator:

asxTUNER_STATUS_VIDEO_VALID Video valid.

asxTUNER_STATUS_VIDEO_COLOR_PRESENT Color present.

asxTUNER_STATUS_VIDEO_IS_60HZ Video is 60 Hz.

asxTUNER_STATUS_VIDEO_HORZ_SYNC_MISSING Horizontal sync is missing.

asxTUNER_STATUS_PLL_LOCKED The tuners PLL is locked onto a signal.

asxTUNER_STATUS_FM_STEREO An FM stereo signal has been detected.

asxTUNER_STATUS_DIGITAL An digital channel has been detected.

asxTUNER_STATUS_MULTIPROGRAM An multi-program channel has been detected.

asxTUNER_STATUS_FIRMWARE_LOADING tuner firmware is loading

9.1.3.31 enum asxTUNERBAND

Tuner band settings.

Enumerator:

asxTUNERBAND_AM Tuner band AM.

asxTUNERBAND_FM Tuner band FM.

asxTUNERBAND_TV Tuner band TV (NTSC North America).

asxTUNERBAND_FM_STEREO Tuner band stereo FM.

asxTUNERBAND_AUX Tuner band auxiliary - input from the 50 pin header.
asxTUNERBAND_TV_PAL_BG Tuner band TV PAL-BG.
asxTUNERBAND_TV_PAL_I Tuner band TV PAL-I.
asxTUNERBAND_TV_PAL_DK Tuner band TV PAL-DK.
asxTUNERBAND_TV_SECAM_L Tuner band TV SECAM.
asxTUNERBAND_DAB

9.1.3.32 enum *asxTUNERDEEMPHASIS*

Tuner FM de-emphasis settings.

Enumerator:

asxTUNERDEEMPHASIS_50 Tuner de-emphasis of 50us (Europe).
asxTUNERDEEMPHASIS_75 Tuner de-emphasis of 75us (USA).
asxTUNERDEEMPHASIS_NONE Tuner no de-emphasis setting.

9.1.3.33 enum *asxTUNERHDBLEND*

Tuner HD Radio blend settings.

Enumerator:

asxTUNERHDBLEND_AUTO Auto blend between analog and digital.
asxTUNERHDBLEND_ANALOG Force analog.

9.1.3.34 enum *asxTUNERMODE*

Tuner mode settings.

Enumerator:

asxTUNERMODE_RSS Tuner mode RSS.
asxTUNERMODE_RSS_ENABLE Tuner mode RSS is enabled.
asxTUNERMODE_RSS_DISABLE Tuner mode RSS is disabled.

9.1.3.35 enum *asxTUNERPROGRAM*

Tuner program settings.

Enumerator:

asxTUNERPROGRAM_NONE Tuner program 1.

asxTUNERPROGRAM_1 Tuner program 1.
asxTUNERPROGRAM_2 Tuner program 2.
asxTUNERPROGRAM_3 Tuner program 3.
asxTUNERPROGRAM_4 Tuner program 4.
asxTUNERPROGRAM_5 Tuner program 5.
asxTUNERPROGRAM_6 Tuner program 6.
asxTUNERPROGRAM_7 Tuner program 7.
asxTUNERPROGRAM_8 Tuner program 8.

9.1.3.36 enum *asxUCONTROL_PFLAGS*

Universal control flags.

Enumerator:

asxPARAM_FLAG_WRITEABLE The control can be set.
asxPARAM_FLAG_READABLE The control can be read (most controls).
asxPARAM_FLAG_VOLATILE The control changes its own value (e.g. meter).

9.1.3.37 enum *asxUCONTROL_PTYPE*

Universal control parameter types.

Enumerator:

asxPARAM_TYPE_NONE
asxPARAM_TYPE_INTEGER
asxPARAM_TYPE_FLOAT
asxPARAM_TYPE_DOUBLE
asxPARAM_TYPE_STRING
asxPARAM_TYPE_IP4_ADDRESS
asxPARAM_TYPE_IP6_ADDRESS
asxPARAM_TYPE_MAC_ADDRESS
asxPARAM_TYPE_BOOLEAN

9.1.3.38 enum asxUCONTROL_RTYPE

Universal control range types.

Enumerator:

asxPARAM_RANGE_NONE
asxPARAM_RANGE_STEPPED_INTEGER
asxPARAM_RANGE_STEPPED_FLOAT
asxPARAM_RANGE_ENUMERATED_INTEGER
asxPARAM_RANGE_ENUMERATED_FLOAT
asxPARAM_RANGE_ENUMERATED
asxPARAM_RANGE_STRING_LENGTH
asxPARAM_RANGE_NUMBER_OF_BITS

9.1.3.39 enum asxVOLUME_AUTOFADE

volume autofade profiles

Enumerator:

asxVOLUME_AUTOFADE_LOG Log fade causes dB level to fade linearly over time.
asxVOLUME_AUTOFADE_LINEAR Linear fade causes amplitude to fade linearly over time.

9.2 asxstring.h File Reference

Defines

- #define [ASX32_API](#)

Functions

- ASX32_API int [ASXSTRING_EnumToString](#) (const int nEnum, char *szString, const int nLength, int *pRequiredLength)
Translate an ASX enum into a string.
- ASX32_API int [ASXSTRING_StringToEnum](#) (const char *szString, int *pnEnum)
Translate a string into an ASX enum.

9.2.1 Define Documentation

9.2.1.1 #define ASX32_API

9.2.2 Function Documentation

9.2.2.1 ASX32_API int ASXSTRING_EnumToString (const int *nEnum*, char * *szString*, const int *nLength*, int * *pRequiredLength*)

Translate an ASX enum into a string.

Parameters

<i>nEnum</i>	The enum value to translate.
<i>szString</i>	The returned string is copied here. The caller should allocate enough memory to hold the returned string. Call this function with <i>szString</i> =0 and check the <i>RequiredLength</i> field to determine how much memory to allocate.
<i>nLength</i>	The length of <i>szString</i> .
<i>pRequiredLength</i>	The required minimum length of the caller's char array.

Returns

Returns 0 on success.

Examples:

[cobranet/main.c](#), [dual_mono_play/main.c](#), [dual_mono_record/main.c](#), [mixer/main.c](#), [mux/main.c](#), [play/main.c](#), [playlist/main.c](#), [tuner/main.c](#), and [volume/main.c](#).

9.2.2.2 ASX32_API int ASXSTRING_StringToEnum (const char * *szString*, int * *pnEnum*)

Translate a string into an ASX enum.

Parameters

<i>szString</i>	The string to look up in enum list.
<i>pnEnum</i>	The enum value returned.

Returns

Returns 0 on success.

9.3 pcxport.txt File Reference

Chapter 10

Example Documentation

10.1 adapter/main.c

This is an example of how to use the ASX Adapter functions.

```
/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/adapter/mai
   n.c,v 1.6 2010/01/11 21:50:31 as-age Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"

int CheckError(ASX_HANDLE hObj, int nLine);

ASX_HANDLE hSystem=0;

int main(int argc, char* argv[])
{
    ASX_ERROR asxError;
    int nAdapters=0;
    int i;

    // create the system
    ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
    CheckError(hSystem, __LINE__);

    // find out how many adapters there are
    asxError = ASX_System_GetAdapterCount(hSystem,&nAdapters);
    CheckError(hSystem, __LINE__);
    printf("There are %d audio adapters in the system \n", nAdapters);

    // loop over the adapters
    for(i=0;i<nAdapters;i++)
    {
        char *pszAdapterName;
        ASX_HANDLE hAdapter;
        unsigned long lSerial;
        char *pszRevision;
        int nLen;
        int nIndex;
        int nDspUtilization;
```

```

    ASX_HANDLE hMixer;

    ASX_System_GetAdapter(hSystem,i,&hAdapter);
    CheckError(hSystem, __LINE__);

    ASX_Adapter_GetName(hAdapter,0,0,&nLen);
    CheckError(hAdapter, __LINE__);
    pszAdapterName = (char *)malloc(nLen);
    ASX_Adapter_GetName(hAdapter,pszAdapterName,nLen,&nLen);
    CheckError(hAdapter, __LINE__);
    printf("Adapter [%d] is %s \n", i,pszAdapterName);

    /* the adapter index is not the same as the loop index */
    ASX_Adapter_GetIndex(hAdapter, &nIndex);
    CheckError(hAdapter, __LINE__);
    printf("Index is %ld \n", nIndex);

    ASX_Adapter_GetSerialNumber(hAdapter,&lSerial);
    CheckError(hAdapter, __LINE__);
    printf("Serial is %ld \n", lSerial);

    pszRevision = (char *)malloc(ASX_SHORT_STRING);
    ASX_Adapter_GetHardwareRevision(hAdapter,pszRevision);
    CheckError(hAdapter, __LINE__);
    printf("Revision is %s \n", pszRevision);

    ASX_Adapter_GetDspUtilization(hAdapter,1,&nDspUtilization);
    CheckError(hAdapter, __LINE__);
    printf("Utilization is %d percent \n", nDspUtilization);

    free(pszAdapterName);
    free(pszRevision);
}

ASX_System_Delete(hSystem);
printf("Press ENTER to exit\n");
getchar();
return 0;
}

int CheckError(ASX_HANDLE hObj, int nLine)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1,nLen2;

    ASX_Error_GetLast( hObj, &nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0,0,&nLen1,0,0,&nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString,nLen1,&nLen1,pszAsxSubSystem
        ErrorString,nLen2,&nLen2);
    printf("Error: %#d, %s - Subsystem Error: %#ld, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
}

```

```

    printf("When called from source %s line %d\n",__FILE__,nLine);

    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

```

10.2 cobranet/main.c

This is an example of how to use the ASX CobraNet functions.

```

/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/cobranet/main.c,v 1.6 2010/01/11 21:50:06 as-age Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "windows.h"
#include "asx.h"
#include "asxstring.h"

int CheckError(ASX_HANDLE hObj, int nLine);
int CheckErrorNonTerminal(ASX_HANDLE hObj, int nLine);
void PrintNodeName(ASX_HANDLE hNode);
void PrintControlName(ASX_HANDLE hControl);
void PrintMeterReadings(ASX_HANDLE hMixer, ASX_HANDLE hControl);
ASX_HANDLE FindSntpCobranetControl(ASX_HANDLE hMixer);

ASX_HANDLE hSystem;

int main(int argc, char* argv[])
{
    char *pszName;
    ASX_HANDLE hAdapter;
    ASX_HANDLE hMixer;
    ASX_HANDLE hNode;
    ASX_HANDLE hControl,hCobraNetControl,hMeterControl;
    ASX_ERROR asxError;
    int nAdapterToUse=0;
    int i,j,nLen,nNodes,nControls,nAdapters;
    unsigned int nBundle;
    unsigned int nMap[8];
    unsigned int nCount;

    // make sure ASX system handle is NULL
    hSystem=NULL;

    // set the address of the host PC network adapter
    ASX_System_SetHostNetworkInterface("192.168.1.106");

    // create the system
    ASX_System_CreateSubSystem(ASX_SYSTEM_TYPE_HPIUDP,&hSystem);
    CheckError(hSystem, __LINE__);

    // add SNMP so that we can control CobraNet devices

```

```

ASX_System_CreateSubSystem(ASX_SYSTEM_TYPE_SNMP, &hSystem);
CheckError(hSystem, __LINE__);

// wait 4 seconds for CobraNet device discovery to complete
printf("Waiting 2 seconds for device discovery to complete.\n");
Sleep(2000);

// list all the adapters
ASX_System_GetAdapterCount(hSystem, &nAdapters);
CheckError(hSystem, __LINE__);
printf("Found %d adapters.\n", nAdapters);

// dump adapter information
for(i=0; i<nAdapters; i++) {
    char szIP[ASX_SHORT_STRING];
    char szInfo[ASX_LONG_STRING];
    int nIndex=0;

    // get the adapter
    asxError = ASX_System_GetAdapter(hSystem, i, &hAdapter);
    CheckError(hSystem, __LINE__);

    ASX_Adapter_GetName(hAdapter, 0, 0, &nLen);
    CheckError(hAdapter, __LINE__);
    pszName = (char *)malloc(nLen);
    ASX_Adapter_GetName(hAdapter, pszName, nLen, &nLen);
    CheckError(hAdapter, __LINE__);
    printf("Adapter [%d] is %s \n", i, pszName);

    /*
    The adapter index is never the same as the loop index.
    Network adapters have a fixed unique index that is independent of there
    IP address or order of discovery.
    */
    ASX_Adapter_GetIndex(hAdapter, &nIndex);
    CheckError(hAdapter, __LINE__);
    printf("\t Index is : %d \n", nIndex);

    ASX_Adapter_GetIpAddress(hAdapter, szIP);
    CheckError(hAdapter, __LINE__);
    printf("\t IP address is : %s \n", szIP);

    // get the mixer handle
    asxError = ASX_Adapter_GetMixer(hAdapter, &hMixer);
    CheckError(hAdapter, __LINE__);

    // get the base control that has some CobraNet stuff
    hCobraNetControl = FindSnmpCobranetControl(hMixer);
    if(hCobraNetControl) {
        asxError = ASX_Cobranet_GetDescription(hCobraNetControl, szInfo, sizeof
(szInfo));
        CheckError(hCobraNetControl, __LINE__);
        if( !asxError )
            printf("\tsysDescription: %s\n", szInfo);
        else
            printf("Error %d\n", asxError);

        asxError = ASX_Cobranet_GetName(hCobraNetControl, szInfo, sizeof(szInfo
));
        CheckError(hCobraNetControl, __LINE__);
        if( !asxError )
            printf("\tsysName: %s\n", szInfo);
    }
}

```

```

        asxError = ASX_Cobranet_GetLocation(hCobraNetControl, szInfo, sizeof(sz
Info));
        CheckError(hCobraNetControl, __LINE__);
        if( !asxError )
            printf("\tsysLocation: %s\n", szInfo);
    } else {
        printf("No CobraNet control on adapter node\n");
    }

}

printf("Enter adapter number to use : ");
scanf("%d", &i);

// get the selected adapter
asxError = ASX_System_GetAdapter(hSystem, i, &hAdapter);
CheckError(hSystem, __LINE__);

// get the mixer handle
asxError = ASX_Adapter_GetMixer( hAdapter, &hMixer );
CheckError(hAdapter, __LINE__);

// get the base control that has some CobraNet stuff
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_COBRANET_IN, 0,
    asxNODE_COBRANET_OUT, 0,
    asxCONTROL_COBRANET, &hCobraNetControl);
CheckError(hMixer, __LINE__);

// dump source lines
ASX_Mixer_GetSourceNodeCount(hMixer, &nNodes);
printf("Source nodes\n");
for(j=0; j<nNodes; j++)
{
    ASX_Mixer_GetSourceNode(hMixer, j, &hNode);
    PrintNodeName(hNode);
    printf("\n");
}
// dump destination lines
ASX_Mixer_GetDestinationNodeCount(hMixer, &nNodes);
printf("Destination nodes\n");
for(j=0; j<nNodes; j++)
{
    ASX_Mixer_GetDestinationNode(hMixer, j, &hNode);
    PrintNodeName(hNode);
    printf("\n");
}

// dump all controls
asxError = ASX_Mixer_GetControlCount(hMixer, &nControls);
CheckError(hMixer, __LINE__);

printf("Retrieved controls (skipping volumes and sample rate converters)\n");

for(i=0; i<nControls; i++)
{
    enum asxCONTROL eControl;
    ASX_Mixer_GetControl(hMixer, i, &hControl);

```

```

    ASX_Control_GetType(hControl, &eControl);
    if( (eControl==asxCONTROL_VOLUME) || (eControl==asxCONTROL_SRC) )
        continue;

    PrintControlName(hControl);

    printf("On node(s) ");
    ASX_Control_GetSourceNode(hControl, &hNode);
    if( hNode )
        PrintNodeName(hNode);
    ASX_Control_GetDestinationNode(hControl, &hNode);
    if( hNode )
        PrintNodeName(hNode);
    printf("\n");
}

// get the details for the first CobraNet transmitter
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    0,0,
    asxNODE_COBRANET_TRANSMITTER,0,
    asxCONTROL_COBRANET_TRANSMITTER, &hCobraNetControl);
CheckError(hMixer, __LINE__);
if(!asxError) {
    ASX_CobranetTx_GetBundle(hCobraNetControl, &nBundle);
    ASX_CobranetTx_GetChannelCount(hCobraNetControl, &nCount);
    ASX_CobranetTx_GetChannelMap(hCobraNetControl, nMap);
    printf("CobraNet transmitter 0 details\n");
    printf("Bundle : %d\n",nBundle);
    printf("Channel count : %d\n",nCount);
    printf("Channel map : %d %d %d %d %d %d %d %d\n",
        nMap[0],
        nMap[1],
        nMap[2],
        nMap[3],
        nMap[4],
        nMap[5],
        nMap[6],
        nMap[7]);
}

// get the details for the first CobraNet receiver
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_COBRANET_RECEIVER,0,
    0,0,
    asxCONTROL_COBRANET_RECEIVER, &hCobraNetControl);
CheckError(hMixer, __LINE__);
if(!asxError) {
    ASX_CobranetRx_GetBundle(hCobraNetControl, &nBundle);
    ASX_CobranetRx_GetChannelMap(hCobraNetControl, nMap);
    printf("CobraNet receiver 0 details\n");
    printf("Bundle : %d\n",nBundle);
    printf("Channel map : %d %d %d %d %d %d %d %d\n",
        nMap[0],
        nMap[1],
        nMap[2],
        nMap[3],
        nMap[4],
        nMap[5],
        nMap[6],
        nMap[7]);
}

```

```

        nMap[7]);
    }

    // look for some peak meters on CobraNet nodes

    // This is a meter on the audio path from a CobraNet Rx that is on the
    // input side (source node) of an ASI2416.
    asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
        hMixer,
        asxNODE_COBRANET_IN, 0,
        asxNODE_NONE, 0,
        asxCONTROL_METER, &hMeterControl);
    CheckError(hMixer, __LINE__);

    if(!asxError) {
        printf("Found meter on first CobraNet input 0 from the network\n");
        PrintMeterReadings(hMixer, hMeterControl);
    }

    // This is a meter on the audio path from the ASI2416 to a CobraNet Tx.
    asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
        hMixer,
        asxNODE_NONE, 0,
        asxNODE_COBRANET_OUT, 0,
        asxCONTROL_METER, &hMeterControl);
    CheckError(hMixer, __LINE__);

    if(!asxError) {
        printf("Found meter on first CobraNet output 0 to the network\n");
        PrintMeterReadings(hMixer, hMeterControl);
    }

    printf("Press ENTER to exit\n");
    getchar();
    ASX_System_Delete(hSystem);
    return 0;
}

void PrintMeterReadings(ASX_HANDLE hMixer, ASX_HANDLE hControl)
{
    int i;
    int chans=0;
    float readings[2];
    ASX_ERROR asxError;

    asxError = ASX_Meter_GetChannels(hControl, (int *)&chans );
    CheckError(hMixer, __LINE__);

    asxError = ASX_Meter_GetPeak(hControl, readings, chans );
    CheckError(hMixer, __LINE__);
    if(!asxError)
        for(i=0;i<chans;i++)
            printf("Meter[%d] reads peak of %5.3f dB\n",i,readings[i]);

    asxError = ASX_Meter_GetRMS(hControl, readings, chans );
    CheckError(hMixer, __LINE__);
    if(!asxError)
        for(i=0;i<chans;i++)
            printf("Meter[%d] reads RMS of %5.3f dB\n",i,readings[i]);
}

```

```

ASX_HANDLE FindSnmpCobranetControl(ASX_HANDLE hMixer)
{
    ASX_HANDLE hSnmpCobranet = NULL;
    ASX_ERROR asxError;
    int nControl, nNumControls;
    enum asxCONTROL eControlType;
    int nSubSystem;

    asxError = ASX_Mixer_GetControlCount(
        hMixer,
        &nNumControls );

    for(nControl=0;nControl < nNumControls;nControl++){
        asxError = ASX_Mixer_GetControl(
            hMixer,
            nControl,
            &hSnmpCobranet);
        if(asxError == asxERROR_NO_ERROR){
            ASX_Control_GetType(hSnmpCobranet,&eControlType);
            ASX_Control_GetSubSystem(hSnmpCobranet,&nSubSystem);
            if(eControlType==asxCONTROL_COBRANET &&
                nSubSystem==ASX_SYSTEM_TYPE_SNMP){
                break;
            }
        }
    }
    if(nControl == nNumControls){
        hSnmpCobranet = NULL;
    }
    return hSnmpCobranet;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;
    int nLen;
    enum asxCONTROL eControl;

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl,0,0,&nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eControl,pszName,nLen,&nLen);
    printf("Control : %s ",pszName);

    free(pszName);
}

void PrintNodeName(ASX_HANDLE hNode)
{
    char *pszName;
    int nLen,nIndex;
    enum asxNODE eNode;

    ASX_Node_GetType(hNode, &eNode);
    ASX_Node_GetIndex(hNode, &nIndex);
    ASXSTRING_EnumToString(eNode,0,0,&nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eNode,pszName,nLen,&nLen);
    printf("Node : %s_%d ",pszName,nIndex);
    free(pszName);
}

```

```

int CheckError(ASX_HANDLE hObj, int nLine)
{
    ASX_ERROR asxError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1,nLen2;

    ASX_Error_GetLast( hObj, &asxError, &asxSubSystemErrorCode);
    if(!asxError)
        return 0;
    ASX_Error_GetLastString( hObj, 0,0,&nLen1,0,0,&nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString,nLen1,&nLen1,pszAsxSubSystem
        ErrorString,nLen2,&nLen2);
    printf("Error: #%d, %s - Subsystem Error: #%ld, %s \n",
        asxError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
    printf("When called from source %s line %d\n",__FILE__,nLine);

    getchar();
    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

int CheckErrorNonTerminal(ASX_HANDLE hObj, int nLine)
{
    ASX_ERROR asxError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1,nLen2;

    ASX_Error_GetLast( hObj, &asxError, &asxSubSystemErrorCode);
    if(!asxError)
        return 0;
    ASX_Error_GetLastString( hObj, 0,0,&nLen1,0,0,&nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString,nLen1,&nLen1,pszAsxSubSystem
        ErrorString,nLen2,&nLen2);
    printf("WARNING: #%d, %s - Skipping.\n\n",
        asxError,
        pszAsxErrorString);
    ASX_Error_Clear( hObj );
    return 1;
}

```

10.3 csharp_asx_player/Form1.cs

This is an example of how to use the ASX Player control from c#.

10.4 dual_mono_play/main.c

This is an example of how to set up channel mode controls for dual mono playback. All play streams on AudioScience audio adapters play, at a minimum, either mono or stereo files on the same device. Mono playback always converts mono or stereo to a stereo stream of audio. So to play to the right channel, for example, the stereo stream should be converted to have only right channel audio output.

Configuration Steps

The following steps should be performed to set up playback of 4 independent mono streams. We are going to play in the following configuration:

```
Play 1 -> Line Out 1 Left
Play 2 -> Line Out 1 Right
Play 3 -> Line Out 2 Left
Play 4 -> Line Out 2 Right
```

Find and set the channel mode controls on Play nodes as follows:

```
Play 1, Channel Mode = "stereo to left" -> Line Out 1 Left
Play 2, Channel Mode = "stereo to right" -> Line Out 1 Right
Play 3, Channel Mode = "stereo to left" -> Line Out 2 Left
Play 4, Channel Mode = "stereo to right" -> Line Out 3 Right
```

The mixer also needs to be adjusted so that Play 1 -> Line Out 1 is set to full volume and Play 2 to Line Out 1 is also set to full volume.

These instructions remain the same whether the functionality is implemented with Microsoft Multimedia waveXXXX() and mixerXXX() calls, HPI or ASX calls.

```
/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/dual_mono_p
   lay/main.c,v 1.2 2009/02/18 20:37:56 as-tfe Exp $ */

/* This examples sets up the playback channel mode controls to support dual mono
   playback.
*/
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"
#include "asxstring.h"

#define MAX_PLAYS 32

ASX_HANDLE hSystem=0;

int CheckError(ASX_HANDLE hObj, int nLine);
void PrintControlName(ASX_HANDLE hControl);

int main(int argc, char* argv[])
{
    char *pszName;
    ASX_HANDLE hAdapter;
```

```

ASX_HANDLE hMixer;
ASX_HANDLE hChannelMode[MAX_PLAYS];
ASX_HANDLE hVolume;
int nAdapterToUse=0;
int i,nLen;
float fVolume[2];

// create the system
ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
CheckError(hSystem, __LINE__);

// get the adapter
ASX_System_GetAdapter(hSystem,nAdapterToUse,&hAdapter);
CheckError(hAdapter, __LINE__);

ASX_Adapter_GetName(hAdapter,0,0,&nLen);
CheckError(hAdapter, __LINE__);
pszName = (char *)malloc(nLen);
ASX_Adapter_GetName(hAdapter,pszName,nLen,&nLen);
CheckError(hAdapter, __LINE__);
printf("Adapter [%d] is %s \n", nAdapterToUse,pszName);

// get the mixer handle
ASX_Adapter_GetMixer(hAdapter, &hMixer );
CheckError(hAdapter, __LINE__);

// Grab a player's channel mode
for(i=0; i<MAX_PLAYS; i++)
{
    // get channel mode object
    ASX_ERROR err = ASX_Mixer_GetControlByNodeTypeAndIndex(
        hMixer,
        asxNODE_PLAYER,i,
        0,0,
        asxCONTROL_CHANNEL_MODE,
        &hChannelMode[i]);
    if(err) // error will be returned when i > number of plays.
        break;

    CheckError(hMixer, __LINE__);

    // set the channel mode
    if((i&1)==0)
    { // even
        ASX_ChannelMode_Set( hChannelMode[i], asxCHANNELMODE_STEREOLEFT);
        CheckError(hChannelMode[i], __LINE__);
    }
    else
    { // odd
        ASX_ChannelMode_Set( hChannelMode[i], asxCHANNELMODE_STEREOTORIGHT);
        CheckError(hChannelMode[i], __LINE__);
    }

    // get volume object
    err = ASX_Mixer_GetControlByNodeTypeAndIndex(
        hMixer,
        asxNODE_PLAYER,i,
        asxNODE_LINE_OUT,i/2,
        asxCONTROL_VOLUME,
        &hVolume);
    if(err) // error will be returned when i > number of plays.

```

```

        break;

        CheckError(hMixer, __LINE__);

        ASX_Volume_GetGain( hVolume, fVolume, 2);
        CheckError(hVolume, __LINE__);
        fVolume[i&1] = 0.0;
        ASX_Volume_SetGain( hVolume, fVolume, 2);
        CheckError(hVolume, __LINE__);
    }

    printf("Press ENTER to exit\n");
    getchar();
    ASX_System_Delete(hSystem);
    return 0;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;
    int nLen;
    enum asxCONTROL eControl;

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl, 0, 0, &nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eControl, pszName, nLen, &nLen);
    printf("Control : %s\n", pszName);

    free(pszName);
}

int CheckError(ASX_HANDLE hObj, int nLine)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1,nLen2;

    ASX_Error_GetLast( hObj, &nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString, nLen1, &nLen1, pszAsxSubSystem
        ErrorString, nLen2, &nLen2);
    printf("Error: #%d, %s - Subsystem Error: #%ld, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
    printf("When called from source %s line %d\n",__FILE__,nLine);

    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

```

```
}
```

10.5 dual_mono_record/main.c

This is an example of how to use the ASX Recorder to record dual mono inputs. All record streams on AudioScience audio adapters record, at a minimum, either mono or stereo files on the same device. Mono recording always records the left channel to an audio file. So to record the right channel of a line in, the channels must be swapped somewhere in the record path. This can be done using the Channel Mode control that is present on the record node of most AudioScience adapters.

Configuration Steps

The following steps should be performed to setup recording of 4 independent mono streams. We are going to record in the following configuration:

```
Record 1 <- Line In 1 Left
Record 2 <- Line In 1 Right
Record 3 <- Line In 2 Left
Record 4 <- Line In 2 Right
```

1. Find multiplexer controls on Record nodes and set sources as follows:

```
Record 1 <- Line In 1
Record 2 <- Line In 1
Record 3 <- Line In 2
Record 4 <- Line In 2
```

2. Find channel mode controls and set up as follows:

```
Record 1, Channel Mode = "normal"
Record 2, Channel Mode = "swap"
Record 3, Channel Mode = "normal"
Record 4, Channel Mode = "swap"
```

3. Open record devices

4. Set record streams to record 4 mono files

5. Start recording.

These instructions remain the same whether the functionality is implemented with Microsoft Multimedia waveXXXX() and mixerXXX() calls, HPI or ASX calls. The next section details ASX recording instructions.

```
/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/dual_mono_r
   ecord/main.c,v 1.5 2004/10/13 18:16:08 as-tfe Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"
#include "asxstring.h"
```

```

ASX_HANDLE hSystem=0;

int CheckError(ASX_HANDLE hObj, int nLine);
void PrintControlName(ASX_HANDLE hControl);

int main(int argc, char* argv[])
{
    char *pszName;
    ASX_HANDLE hAdapter;
    ASX_HANDLE hMixer;
    ASX_HANDLE hRecorder[4];
    ASX_HANDLE hChannelMode[4];
    ASX_HANDLE hMux[4];
    int nAdapterToUse=0;
    int i,nLen;
    char szFileNames[][16]={ "test1.wav","test2.wav","test3.wav","test4.wav"};

    // create the system
    ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
    CheckError(hSystem, __LINE__);

    // get the adapter
    ASX_System_GetAdapter(hSystem,nAdapterToUse,&hAdapter);
    CheckError(hAdapter, __LINE__);

    ASX_Adapter_GetName(hAdapter,0,0,&nLen);
    CheckError(hAdapter, __LINE__);
    pszName = (char *)malloc(nLen);
    ASX_Adapter_GetName(hAdapter,pszName,nLen,&nLen);
    CheckError(hAdapter, __LINE__);
    printf("Adapter [%d] is %s \n", nAdapterToUse,pszName);

    // get the mixer handle
    ASX_Adapter_GetMixer( hAdapter, &hMixer );
    CheckError(hAdapter, __LINE__);

    // Grab a recorder, channel mode and mux control for each recorder.
    for(i=0; i<4; i++)
    {
        // get a record object
        ASX_Mixer_GetControlByNodeTypeAndIndex(
            hMixer,
            0,0,
            asxNODE_RECORDER,i,
            asxCONTROL_RECORDER,
            &hRecorder[i]);
        CheckError(hMixer, __LINE__);

        // print out some control details
        PrintControlName(hRecorder[i]);

        // get multiplexer object
        ASX_Mixer_GetControlByNodeTypeAndIndex(
            hMixer,
            0,0,
            asxNODE_RECORDER,i,
            asxCONTROL_MULTIPLEXER,
            &hMux[i]);
        CheckError(hMixer, __LINE__);

        // set the multiplexer to the correct line in
        switch(i)

```

```

    {
        case 0:
        case 1:
            ASX_Multiplexer_Set( hMux[i], asxNODE_LINE_IN, 0);
            CheckError(hMux[i], __LINE__);
            break;
        case 2:
        case 3:
            ASX_Multiplexer_Set( hMux[i], asxNODE_LINE_IN, 1);
            CheckError(hMux[i], __LINE__);
            break;
    }

    // get channel mode object
    ASX_Mixer_GetControlByNodeTypeAndIndex(
        hMixer,
        0, 0,
        asxNODE_RECORDER, i,
        asxCONTROL_CHANNEL_MODE,
        &hChannelMode[i]);
    CheckError(hMixer, __LINE__);

    // set the channel mode
    switch(i)
    {
        case 0:
        case 2:
            ASX_ChannelMode_Set( hChannelMode[i], asxCHANNELMODE_NORMAL);
            CheckError(hChannelMode[i], __LINE__);
            break;
        case 1:
        case 3:
            ASX_ChannelMode_Set( hChannelMode[i], asxCHANNELMODE_SWAP);
            CheckError(hChannelMode[i], __LINE__);
            break;
    }
}

// open the player and pass in the file to be played
for(i=0; i<4; i++)
{
    printf("Open recorder %d\n", i);
    ASX_Recorder_Open(
        hRecorder[i],
        szFileNames[i],
        asxFILE_FORMAT_WAV,          // file format
        asxFILE_MODE_CREATE,        // file mode (create vs append)
        1,                          // channels = 1 or 2 at present
        asxAUDIO_FORMAT_PCM16,      // sample format
        44100,                      // sample rate = 8000 to 192000 Hz
        0,                          // 8000 to 384000 bps (MPEG only)
        asxRECORD_MODE_DONT_CARE    // MPEG mode
    );
    CheckError(hRecorder[i], __LINE__);
}

// start recording all files
for(i=0; i<4; i++)
{
    ASX_Recorder_Start( hRecorder[i] );
    CheckError(hRecorder[i], __LINE__);
}

```

```

    // wait for recording completion
    printf("Hit enter to end recording\n");
    getchar();

    for(i=0; i<4; i++)
    {
        ASX_Recorder_Stop( hRecorder[i] );
        CheckError(hRecorder[i], __LINE__);
    }

    printf("Recording complete.\n");

    // close the file being recorded
    for(i=0; i<4; i++)
    {
        ASX_Recorder_Close(hRecorder[i]);
        CheckError(hRecorder[i], __LINE__);
    }

    printf("Press ENTER to exit\n");
    getchar();
    ASX_System_Delete(hSystem);
    return 0;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;
    int nLen;
    enum asxCONTROL eControl;

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl, 0, 0, &nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eControl, pszName, nLen, &nLen);
    printf("Control : %s\n", pszName);

    free(pszName);
}

int CheckError(ASX_HANDLE hObj, int nLine)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1, nLen2;

    ASX_Error_GetLast( hObj, &nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString, nLen1, &nLen1, pszAsxSubSystem
        ErrorString, nLen2, &nLen2);
    printf("Error: %#d, %s - Subsystem Error: %#ld, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
}

```

```

    printf("When called from source %s line %d\n", __FILE__, nLine);

    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

```

10.6 mixer/main.c

This is an example of how to use the ASX Mixer functions.

```

/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/mixer/main.
   c,v 1.8 2011/03/01 14:26:48 as-age Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"
#include "asxstring.h"

int CheckError(ASX_HANDLE hObj, int nLine);
int CheckErrorNonTerminal(ASX_HANDLE hObj, int nLine);
void PrintNodeName(ASX_HANDLE hNode);
void PrintControlName(ASX_HANDLE hControl);

ASX_HANDLE hSystem=0;

int main(int argc, char* argv[])
{
    char *pszName;
    char szString[256];
    ASX_HANDLE hAdapter;
    ASX_HANDLE hMixer;
    ASX_HANDLE hNode, hSrcNode, hDestNode;
    ASX_HANDLE hControl;
    ASX_ERROR asxError;
    int nAdapterToUse=0;
    int i, j, nLen, nNodes, nControls;

    // create the system
    ASX_System_Create(ASX_SYSTEM_TYPE_HPI, &hSystem);
    CheckError(hSystem, __LINE__);

    // get the adapter
    asxError = ASX_System_GetAdapter(hSystem, nAdapterToUse, &hAdapter);
    CheckError(hSystem, __LINE__);

    ASX_Adapter_GetName(hAdapter, 0, 0, &nLen);
    CheckError(hAdapter, __LINE__);
    pszName = (char *)malloc(nLen);
    ASX_Adapter_GetName(hAdapter, pszName, nLen, &nLen);
    CheckError(hAdapter, __LINE__);
    printf("Adapter [%d] is %s \n", nAdapterToUse, pszName);

    // get the MAC address (only supported on some adapters)

```

```

asxError = ASX_Adapter_GetMacAddress(hAdapter,szString);
if (asxError==asxERROR_NO_ERROR)
{
    printf("\tMAC address is %s \n", szString);
}

// get the mixer handle
asxError = ASX_Adapter_GetMixer( hAdapter, &hMixer );
CheckError(hAdapter, __LINE__);

// dump source lines
ASX_Mixer_GetSourceNodeCount(hMixer,&nNodes);
printf("Source nodes\n");
for(j=0;j<nNodes;j++)
{
    ASX_Mixer_GetSourceNode(hMixer,j,&hNode);
    PrintNodeName(hNode);
}
// dump destination lines
ASX_Mixer_GetDestinationNodeCount(hMixer,&nNodes);
printf("Destination nodes\n");
for(j=0;j<nNodes;j++)
{
    ASX_Mixer_GetDestinationNode(hMixer,j,&hNode);
    PrintNodeName(hNode);
}

// find all LineIn nodes (for exmaple)
asxError = ASX_Mixer_GetNodeTypeCount(hMixer,asxNODE_LINE_IN,&nNodes);
CheckError(hMixer, __LINE__);
printf("Total of %d asxNODE_LINE_IN nodes found.\n",nNodes);

for(i=0;i<nNodes;i++)
{
    asxError = ASX_Mixer_GetNodeByType(hMixer,asxNODE_LINE_IN,i,&hNode);
    CheckError(hMixer, __LINE__);
    PrintNodeName(hNode);
}

// dump all controls
asxError = ASX_Mixer_GetControlCount(hMixer,&nControls);
CheckError(hMixer, __LINE__);

printf("Retrieved controls\n");
for(i=0;i<nControls;i++)
{
    ASX_Mixer_GetControl(hMixer,i,&hControl);
    PrintControlName(hControl);

    printf("On node(s) ");
    ASX_Control_GetSourceNode(hControl,&hNode);
    if( hNode )
        PrintNodeName(hNode);
    ASX_Control_GetDestinationNode(hControl,&hNode);
    if( hNode )
        PrintNodeName(hNode);
    printf("\n");
}

// ***** Using ASX_Mixer_GetControlByNode()

```

```

printf("ASX_Mixer_GetControlByNode() examples\n");

// ----- get a peak meter on node Play 0
printf("Finding a peak meter control of node Player 0\n");
// first get the Player node
asxError = ASX_Mixer_GetNodeByType(hMixer, asxNODE_PLAYER, 0, &hNode);
if(!CheckErrorNonTerminal(hMixer, __LINE__))
{
    // now get the control
    asxError = ASX_Mixer_GetControlByNode(hMixer, hNode, 0, asxCONTROL_METER, &hControl);
    if(!CheckErrorNonTerminal(hMixer, __LINE__))
    {
        // print out some controldetails
        PrintControlName(hControl);
    }
}

// ----- get a trim/level control on Line Out 0
printf("Finding a level/trim control of node Line Out 0\n");
// first get the Line Out node
asxError = ASX_Mixer_GetNodeByType(hMixer, asxNODE_LINE_OUT, 0, &hNode);
if(!CheckErrorNonTerminal(hMixer, __LINE__))
{
    // now get the control
    asxError = ASX_Mixer_GetControlByNode(hMixer, 0, hNode, asxCONTROL_LEVEL, &hControl);
    if(!CheckErrorNonTerminal(hMixer, __LINE__))
    {
        // print out some controldetails
        PrintControlName(hControl);
    }
}

// ----- get a volume control between Play 0 and Line Out 0
printf("Finding a volume control between Play 0 and Line Out 0\n");
// first get the Line Out destination node
asxError = ASX_Mixer_GetNodeByType(hMixer, asxNODE_LINE_OUT, 0, &hDestNode);
if(!CheckErrorNonTerminal(hMixer, __LINE__))
{
    // second get the Play source node node
    asxError = ASX_Mixer_GetNodeByType(hMixer, asxNODE_PLAYER, 0, &hSrcNode);
    if(!CheckErrorNonTerminal(hMixer, __LINE__))
    {
        // now get the control
        asxError = ASX_Mixer_GetControlByNode(hMixer, hSrcNode, hDestNode, asxCONTROL_VOLUME, &hControl);
        if(!CheckErrorNonTerminal(hMixer, __LINE__))
        {
            // print out some controldetails
            PrintControlName(hControl);
        }
    }
}

// ***** Using ASX_Mixer_GetControlByNodeTypeAndIndex()
printf("ASX_Mixer_GetControlByNodeTypeAndIndex() examples\n");
// ----- get a peak meter on node Play 0
printf("Finding a peak meter control of node Player 0\n");
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_PLAYER, 0,

```

```

        0,0,
        asxCONTROL_METER,
        &hControl);
if(!CheckErrorNonTerminal(hMixer, __LINE__))
{
    // print out some control details
    PrintControlName(hControl);
}

// ----- get a trim/level control on Line Out 0
printf("Finding a level/trim control of node Line Out 0\n");
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    0,0,
    asxNODE_LINE_OUT,0,
    asxCONTROL_LEVEL,
    &hControl);
if(!CheckErrorNonTerminal(hMixer, __LINE__))
{
    // print out some control details
    PrintControlName(hControl);
}

// ----- get a volume control between Play 0 and Line Out 0
printf("Finding a volume control between Play 0 and Line Out 0\n");
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_PLAYER,0,
    asxNODE_LINE_OUT,0,
    asxCONTROL_VOLUME,
    &hControl);
if(!CheckErrorNonTerminal(hMixer, __LINE__))
{
    // print out some control details
    PrintControlName(hControl);
}

printf("Press ENTER to exit\n");
getchar();
ASX_System_Delete(hSystem);
return 0;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;
    int nLen;
    enum asxCONTROL eControl;

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl,0,0,&nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eControl,pszName,nLen,&nLen);
    printf("Control : %s\n",pszName);

    free(pszName);
}

void PrintNodeName(ASX_HANDLE hNode)
{
    char *pszName;
    int nLen,nIndex;

```

```

enum asxNODE eNode;

ASX_Node_GetType(hNode, &eNode);
ASX_Node_GetIndex(hNode, &nIndex);
ASXSTRING_EnumToString(eNode, 0, 0, &nLen);
pszName=(char *)malloc(nLen);
ASXSTRING_EnumToString(eNode, pszName, nLen, &nLen);
printf("Node : %s_%d\n", pszName, nIndex);
free(pszName);
}

int CheckError(ASX_HANDLE hObj, int nLine)
{
    ASX_ERROR nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1, nLen2;

    ASX_Error_GetLast( hObj, &nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString, nLen1, &nLen1, pszAsxSubSystem
        ErrorString, nLen2, &nLen2);
    printf("Error: #%d, %s - Subsystem Error: #%ld, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
    printf("When called from source %s line %d\n", __FILE__, nLine);

    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

int CheckErrorNonTerminal(ASX_HANDLE hObj, int nLine)
{
    ASX_ERROR nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1, nLen2;

    ASX_Error_GetLast( hObj, &nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString, nLen1, &nLen1, pszAsxSubSystem
        ErrorString, nLen2, &nLen2);
    printf("WARNING: #%d, %s - Skipping.\n\n",
        nError,
        pszAsxErrorString);

```

```
    ASX_Error_Clear( hObj );
    return 1;
}
```

10.7 mux/main.c

This is an example of how to use the ASX multiplexer control.

```
/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/mux/main.c,
   v 1.5 2004/08/11 14:13:36 as-age Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"
#include "asxstring.h"

int CheckError(ASX_HANDLE hObj, int nLine);
void PrintControlName(ASX_HANDLE hControl);

ASX_HANDLE hSystem=0;

int main(int argc, char* argv[])
{
    char *pszName;
    ASX_HANDLE hAdapter;
    ASX_HANDLE hMixer;
    ASX_HANDLE hMux;
    ASX_ERROR asxError;
    int nAdapterToUse=0;
    enum asxNODE eNode;
    int nNodeIndex;
    char szName[64];
    int nRequiredLength;
    int nCount;
    int i,nLen;

    // create the system
    ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
    CheckError(hSystem, __LINE__);

    // get the adapter
    asxError = ASX_System_GetAdapter(hSystem,nAdapterToUse,&hAdapter);
    CheckError(hSystem, __LINE__);

    ASX_Adapter_GetName(hAdapter,0,0,&nLen);
    CheckError(hAdapter, __LINE__);
    pszName = (char *)malloc(nLen);
    ASX_Adapter_GetName(hAdapter,pszName,nLen,&nLen);
    CheckError(hAdapter, __LINE__);
    printf("Adapter [%d] is %s \n", nAdapterToUse,pszName);

    // get the mixer handle
    asxError = ASX_Adapter_GetMixer( hAdapter, &hMixer );
    CheckError(hAdapter, __LINE__);

    // get a multiplexer object
    asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
        hMixer,
        0,0,
        asxNODE_RECORDER,0,
```

```

        asxCONTROL_MULTIPLEXER,
        &hMux);
    CheckError(hMixer, __LINE__);
    // print out some control details
    PrintControlName(hMux);

    // Get the selector and print out available selections,
    asxError=0;
    i=0;
    while(!asxError)
    {

        asxError=ASX_Multiplexer_Enumerate( hMux, i,&eNode, &nNodeIndex, &nCount)
    ;
        if(i==0)
            printf("Multiplexer has total of %d options\n",nCount);

        if(!asxError)
        {
            ASXSTRING_EnumToString(eNode, szName, 64, &nRequiredLength);
            printf("Option[%d] is %s %d\n",i,szName,nNodeIndex);
            i++;
        }
    }

    // Get the current setting.
    asxError = ASX_Multiplexer_Get(hMux,&eNode,&nNodeIndex);
    ASXSTRING_EnumToString(eNode, szName, 64, &nRequiredLength);
    printf("Multiplexer currently set to enum %d, index %d, %s %d\n",eNode,nNodeIndex,szName,nNodeIndex);

    printf("Press ENTER to exit\n");
    getchar();
    ASX_System_Delete(hSystem);
    return 0;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;
    int nLen;
    enum asxCONTROL eControl;

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl,0,0,&nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eControl,pszName,nLen,&nLen);
    printf("Control : %s\n",pszName);

    free(pszName);
}

int CheckError(ASX_HANDLE hObj, int nLine)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1,nLen2;

    ASX_Error_GetLast( hObj, &nError, &asxSubSystemErrorCode);

```

```

    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0,0,&nLen1,0,0,&nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString,nLen1,&nLen1,pszAsxSubSystem
        ErrorString,nLen2,&nLen2);
    printf("Error: #%d, %s - Subsystem Error: #%ld, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
    printf("When called from source %s line %d\n",__FILE__,nLine);

    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

```

10.8 play/main.c

This is an example of how to use the ASX Player control.

```

/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/play/main.c
   ,v 1.14 2011/05/04 19:13:02 as-dxb Exp $ */
#include "stdio.h"
#include "stdlib.h"
#ifdef __APPLE__
#include <unistd.h>
#include <asx/asx.h>
#include <asx/asxstring.h>
#define Sleep(t) sleep(t)
#else
#include "windows.h"
#include "asx.h"
#include "asxstring.h"
#endif

ASX_HANDLE hSystem=0;

void PrintControlName(ASX_HANDLE hControl);
int CheckError(ASX_HANDLE hObj, int nLine);

int main(int argc, char* argv[])
{
    char *pszName;
    char *pszFormat;
    char *pDummy;
    char *pState;
    ASX_HANDLE hAdapter;
    ASX_HANDLE hMixer;
    ASX_HANDLE hPlayer;
    ASX_ERROR asxError;
    int nAdapterToUse=0;
    int nLen;

```

```

enum asxPLAYER_STATE state;

if(argc<2)
{
    printf("Filename to play is a required parameter.\n");
    return -1;
}

// create the system
ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
CheckError(hSystem, __LINE__);

ASX_System_SetMessageLogging(hSystem,asxMSG_LOGGING_VERBOSE);
CheckError(hSystem, __LINE__);

// get the adapter
ASX_System_GetAdapter(hSystem,nAdapterToUse,&hAdapter);
CheckError(hAdapter, __LINE__);

ASX_Adapter_GetName(hAdapter,0,0,&nLen);
CheckError(hAdapter, __LINE__);
pszName = (char *)malloc(nLen);
ASX_Adapter_GetName(hAdapter,pszName,nLen,&nLen);
CheckError(hAdapter, __LINE__);
printf("Adapter [%d] is %s \n", nAdapterToUse,pszName);

// get the mixer handle
ASX_Adapter_GetMixer( hAdapter, &hMixer );
CheckError(hAdapter, __LINE__);

// get a player object
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_PLAYER,0,
    0,0,
    asxCONTROL_PLAYER,
    &hPlayer);
CheckError(hMixer, __LINE__);

// print out some control details
PrintControlName(hPlayer);

// open the player and pass in the file to be played
ASX_Player_Open( hPlayer, argv[1]);
CheckError(hPlayer, __LINE__);

if(argc>2)
{
    printf("Start at offset %s milliseconds\n",argv[2]);
    ASX_Player_PreLoad(hPlayer,asxTIMESCALE_MILLISECONDS,strtoul(argv[2],&pDum
my,10));
}

// start playing the file at offset 0 seconds.
ASX_Player_Start( hPlayer );
CheckError(hPlayer, __LINE__);

ASX_Player_Format_GetString(hPlayer, &pszFormat);
printf("Playing %s Format %s on Device %s\n",argv[1],pszFormat,pszName);
free(pszName);

#define TEST 1

```

```

#if TEST==0
    printf("Waiting for playback to complete\n");
    ASX_Player_Wait(hPlayer);
    CheckError(hPlayer, __LINE__);
#elif TEST==1
    printf("Looping while player state is asxPLAYER_RUNNING\n");
    while(1){
        enum asxPLAYER_STATE s;
        ASX_Player_GetState(hPlayer, &s);
        CheckError(hPlayer, __LINE__);
        if(s!=asxPLAYER_RUNNING) {
            printf("Player state %d\n", (int)s);
            break;
        }
        Sleep(10);
    }
#elif TEST==2
    printf("Pausing 1 second before issuing ASX_Player_Stop()\n");
    Sleep(1);
    printf("Call ASX_Player_Stop()\n");
    ASX_Player_Stop(hPlayer);
#else
#error Bad TEST define value.
#endif

    ASX_Player_GetState(hPlayer, &state);
    ASXSTRING_EnumToString(state, 0, 0, &nLen);
    pState = (char *)malloc(nLen);
    ASXSTRING_EnumToString(state, pState, nLen, &nLen);
    printf("Playback complete. Final state:%s\n", pState);

    // close the file being played
    ASX_Player_Close(hPlayer);
    CheckError(hPlayer, __LINE__);

/*

Not tested....

// start position other than 0
ASX32_API ASX_ERROR ASX_Player_Start( ASX_HANDLE hPlayer, float fPosition);
ASX32_API ASX_ERROR ASX_Player_Pause( ASX_HANDLE hPlayer);
ASX32_API ASX_ERROR ASX_Player_Stop( ASX_HANDLE hPlayer);
ASX32_API ASX_ERROR ASX_Player_GetPosition( ASX_HANDLE hPlayer, float *pfPosition
);
ASX32_API ASX_ERROR ASX_Player_GetState( ASX_HANDLE hPlayer, int *pstate);
ASX32_API ASX_ERROR ASX_Player_SetTimeScale( ASX_HANDLE hPlayer, float fScaleFact
or);
*/

    printf("Press ENTER to exit\n");
    getchar();
    ASX_System_Delete(hSystem);
    return 0;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;
    int nLen;
    enum asxCONTROL eControl;

```

```

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl, 0, 0, &nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eControl, pszName, nLen, &nLen);
    printf("Control : %s\n", pszName);

    free(pszName);
}
int CheckError(ASX_HANDLE hObj, int nLine)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1, nLen2;

    ASX_Error_GetLast( hObj, (ASX_ERROR*)&nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString, nLen1, &nLen1, pszAsxSubSystem
        ErrorString, nLen2, &nLen2);
    printf("Error: #d, %s - Subsystem Error: #d, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
    printf("When called from source %s line %d\n", __FILE__, nLine);

    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

```

10.9 playlist/main.c

This is an example of how to use the ASX Player to play filelists.

```

/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/playlist/main.c,v 1.6 2010/01/19 14:49:46 as-tfe Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"
#include "asxstring.h"

ASX_HANDLE hSystem=0;
int list_done = 0;

void PrintControlName(ASX_HANDLE hControl);
int CheckError(ASX_HANDLE hObj, int nLine);

void PlayerCallback

```

```

(
    ASX_HANDLE hASX_Player_Object,
    const enum asxPLAYER_FLAGS flags,
    void *pUser1
)
{
    if(flags & asxPLAYER_FILE_COMPLETE){
        printf("File done callback\n");
    }
    if(flags & asxPLAYER_FILELIST_COMPLETE){
        printf("File list done callback\n");
        list_done = 1;
    }
    if(flags & asxPLAYER_FILE_START){
        char *szCurrentFilename = NULL;
        ASX_Player_PlaylistStatus( hASX_Player_Object,
                                NULL,
                                NULL,
                                &szCurrentFilename,
                                NULL,
                                NULL );
        printf("File %s started\n",szCurrentFilename);
    }
}

void ShowStatus
(
    ASX_HANDLE hASX_Player_Object
)
{
    unsigned int nTotalFileCount = 0;
    int nCurrentFile = 0;
    char *szCurrentFilename = NULL;
    unsigned int nTotalTime_ms = 0;
    unsigned int nCurrentTime_ms = 0;

    static unsigned int nLastTime_ms = 0;

    ASX_Player_PlaylistStatus( hASX_Player_Object,
                              &nTotalFileCount,
                              &nCurrentFile,
                              &szCurrentFilename,
                              &nTotalTime_ms,
                              &nCurrentTime_ms );

    if(nCurrentTime_ms - nLastTime_ms > 50){
        printf("Playing \"%s\" (file %d of %d) pos %f of %f\n",
              szCurrentFilename,nCurrentFile+1,nTotalFileCount,
              (float)nCurrentTime_ms/1000.0, (float)nTotalTime_ms/1000.0);
        nLastTime_ms = nCurrentTime_ms;
    }
}

char *m_buf = NULL;
char **m_list = NULL;
int m_count = 0;

ASX_ERROR OpenPlaylist
(
    ASX_HANDLE hPlayer,
    const char* pFilename
)

```

```

{
    ASX_ERROR result = asxERROR_PLAYER_NOFILE;
    int i, j;
    fpos_t len;
    FILE *fp;

    fp = fopen(pFilename, "r");
    if (!fp)
        return result;
    fseek(fp, 0, SEEK_END);
    fgetpos(fp, &len);
    fseek(fp, 0, SEEK_SET);

    if (len) {
        m_buf = (char *)malloc(len+1);
        fread(m_buf, 1, len, fp);
        m_count=1;
        for(i=0; i<len; i++) {
            if(m_buf[i]=='\n')
                m_count++;
        }
        m_list = (char **)malloc(sizeof(char *)*m_count);
        j = 0;
        m_list[0] = m_buf;
        for(i=0; i<len; i++) {
            if(m_buf[i]=='\n') {
                j++;
                m_buf[i]=0;
                if(j<m_count)
                    m_list[j] = m_buf + i + 1;
            }
        }
        m_buf[len] = 0;
        // Skip leading whitespace and remove empty strings
        for(j=0; j<m_count; j++) {
            while(m_list[j][0]==' ' || m_list[j][0]=='\t')
                m_list[j]++;
            if(m_list[j][0]==0) {
                for(i=j; i<m_count-1; i++)
                    m_list[i] = m_list[i+1];
                m_count--;
                j--;
            }
        }
        result = ASX_Player_OpenPlaylist(hPlayer, m_list, m_count);
    }
    fclose(fp);

    return result;
}

void FreePlaylist(void)
{
    if(m_buf)
        free(m_buf);
    if(m_list)
        free(m_list);
}

int main(int argc, char* argv[])
{
    char *pszName;

```

```

char *pszFormat;
char *pDummy;
char *pExt;
ASX_HANDLE hAdapter;
ASX_HANDLE hMixer;
ASX_HANDLE hPlayer;
ASX_ERROR asxError;
int nAdapterToUse=0;
int nLen;

if(argc<2)
{
    printf( "Filename to play is a required parameter.\n"
           "Filename should be either an audio file (i.e. .WAV or .MP3) or a pla
in text\n"
           "file (with a .TXT extension) containing a list of filenames, one per
line.\n");
    return -1;
}

// create the system
ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
CheckError(hSystem, __LINE__);

// get the adapter
ASX_System_GetAdapter(hSystem,nAdapterToUse,&hAdapter);
CheckError(hSystem, __LINE__);

ASX_Adapter_GetName(hAdapter,0,0,&nLen);
CheckError(hAdapter, __LINE__);
pszName = (char *)malloc(nLen);
ASX_Adapter_GetName(hAdapter,pszName,nLen,&nLen);
CheckError(hAdapter, __LINE__);
printf("Adapter [%d] is %s \n", nAdapterToUse,pszName);

// get the mixer handle
ASX_Adapter_GetMixer( hAdapter, &hMixer );
CheckError(hAdapter, __LINE__);

// get a player object
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_PLAYER, (argc>3)?strtol(argv[3],&pDummy,10):0,
    0,0,
    asxCONTROL_PLAYER,
    &hPlayer);
CheckError(hMixer, __LINE__);

// print out some control details
PrintControlName(hPlayer);

// open the player and pass in the file to be played
// check for '.txt' extension
pExt = strrchr(argv[1],'.');
if(_stricmp(pExt,".txt")==0){
    OpenPlaylist( hPlayer, argv[1]);
}else{
    ASX_Player_Open( hPlayer, argv[1]);
}
CheckError(hPlayer, __LINE__);

ASX_Player_RegisterCallback(hPlayer,PlayerCallback,

```

```

asxPLAYER_FILE_COMPLETE | asxPLAYER_FILELIST_COMPLETE | asxPLAYER_FILE_START,
    NULL);

if(argc>2)
{
    printf("Start at offset %s milliseconds\n",argv[2]);
    ASX_Player_PreLoad(hPlayer,asxTIMESCALE_MILLISECONDS,strtoul(argv[2],&pDum
my,10));
}

// start playing the file at offset 0 seconds.
ASX_Player_Start( hPlayer );
CheckError(hPlayer, __LINE__);

ASX_Player_Format_GetString(hPlayer, &pszFormat);
printf("Playing %s Format %s on Device %s\n",argv[1],pszFormat,pszName);
free(pszName);

// wait for playback completion
printf("Waiting for playback to complete\n");
//while(!list_done)
// ShowStatus(hPlayer);
ASX_Player_PlaylistWait(hPlayer);
CheckError(hPlayer, __LINE__);

printf("Playback complete.\n");

// close the file being played
ASX_Player_Close(hPlayer);
CheckError(hPlayer, __LINE__);

/*

Not tested....

// start position other than 0
ASX32_API ASX_ERROR ASX_Player_Start( ASX_HANDLE hPlayer, float fPosition);
ASX32_API ASX_ERROR ASX_Player_Pause( ASX_HANDLE hPlayer);
ASX32_API ASX_ERROR ASX_Player_Stop( ASX_HANDLE hPlayer);
ASX32_API ASX_ERROR ASX_Player_GetPosition( ASX_HANDLE hPlayer, float *pfPosition
);
ASX32_API ASX_ERROR ASX_Player_GetState( ASX_HANDLE hPlayer, int *pnState);
ASX32_API ASX_ERROR ASX_Player_SetTimeScale( ASX_HANDLE hPlayer, float fScaleFact
or);
*/

printf("Press ENTER to exit\n");
getchar();
FreePlaylist();
ASX_System_Delete(hSystem);
return 0;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;
    int nLen;
    enum asxCONTROL eControl;

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl,0,0,&nLen);
    pszName=(char *)malloc(nLen);

```

```

    ASXSTRING_EnumToString(eControl, pszName, nLen, &nLen);
    printf("Control : %s\n", pszName);

    free(pszName);
}
int CheckError(ASX_HANDLE hObj, int nLine)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1, nLen2;

    ASX_Error_GetLast( hObj, (ASX_ERROR*)&nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString, nLen1, &nLen1, pszAsxSubSystem
        ErrorString, nLen2, &nLen2);
    printf("Error: #%d, %s - Subsystem Error: #%ld, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
    printf("When called from source %s line %d\n", __FILE__, nLine);

    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

```

10.10 record/main.c

This is an example of how to use the ASX Recorder control.

```

/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/record/main
.c,v 1.9 2011/05/04 19:13:02 as-dxb Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"

ASX_HANDLE hSystem=0;

int CheckError(ASX_HANDLE hObj, int nLine);

int main(int argc, char* argv[])
{
    ASX_HANDLE hAdapter;
    ASX_HANDLE hMixer;
    ASX_HANDLE hRecorder;
    ASX_ERROR asxError;
    enum asxFILE_MODE eFileMode = asxFILE_MODE_CREATE;
    int nAdapterToUse=0;

```

```

int c;
int nPaused = 0;
int nKeepGoing = 1;

if(argc<2)
{
    printf("Filename to record is a required parameter.\n");
    return -1;
}
if( (argv[1][0] == '-' || argv[1][0] == '/') &&
    (argv[1][1] == 'a' || argv[1][1] == 'A') )
{
    eFileMode = asxFILE_MODE_APPEND;
    if(argc<3)
    {
        printf("Filename to record is a required parameter.\n");
        return -1;
    }
}

// create the system
ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
CheckError(hSystem, __LINE__);

// get the adapter
ASX_System_GetAdapter(hSystem,nAdapterToUse,&hAdapter);
CheckError(hAdapter, __LINE__);

// get the mixer handle
ASX_Adapter_GetMixer( hAdapter, &hMixer );
CheckError(hAdapter, __LINE__);

// get a recorder object
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_NONE,0,
    asxNODE_RECORDER,0,
    asxCONTROL_RECORDER,
    &hRecorder);
CheckError(hMixer, __LINE__);

// open the player and pass in the file to be played
ASX_Recorder_Open(
    hRecorder, argv[argc-1],
    asxFILE_FORMAT_WAV, // file format = asxFILE_FORMAT_WAV, _MP3, _RAW
    eFileMode, // file mode = asxFILE_MODE_CREATE or _APPEND
    2, // channels = 1 or 2 at present
    asxAUDIO_FORMAT_PCM16, // audio format = asxAUDIO_FORMAT_PCM8, _PCM16, _
    PCM24, _PCM32, _PCM32_FLOAT, _MPEG_L2, _MPEG_L3, _DOLBY_AC2, _MPEG_AACPLUS
    44100, // sample rate = 8 to 192000 Hz
    0, // bitrate = 8000 to 384000 bps (MPEG only)
    asxRECORD_MODE_STEREO // asxRECORD_MODE_STEREO, _JOINT_STEREO, _DUAL_MO
    NO
);
CheckError(hRecorder, __LINE__);

// start playing the file at offset 0 seconds.
ASX_Recorder_Start( hRecorder );
CheckError(hRecorder, __LINE__);

// wait for record completion
while(nKeepGoing){

```

```

        if(nPaused)
            printf("Press \'r\' to resume or \'x\' to end recording.\nCommand: ")
        ;
        else
            printf("Press \'p\' to pause or \'x\' to end recording.\nCommand: ");

        c = getchar();
        // get the rest of the line
        while(getchar()!='\n');
        switch(c){
        case 'p':
            if(nPaused)
            {
                printf("Invalid command.\n");
            }
            else
            {
                ASX_Recorder_Pause( hRecorder );
                nPaused = 1;
            }
            break;
        case 'r':
            if(!nPaused)
            {
                printf("Invalid command.\n");
            }
            else
            {
                ASX_Recorder_Start( hRecorder );
                nPaused = 0;
            }
            break;
        case 'x':
            nKeepGoing = 0;
            break;
        default:
            printf("Invalid command.\n");
            break;
        }
    }

    ASX_Recorder_Stop( hRecorder );
    CheckError(hRecorder, __LINE__);

    printf("Recording complete.\n");

    // close the file being played
    ASX_Recorder_Close(hRecorder);
    CheckError(hRecorder, __LINE__);

    printf("Press ENTER to exit\n");
    getchar();
    ASX_System_Delete(hSystem);
    return 0;
}

int CheckError(ASX_HANDLE hObj, int nLine)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;

```

```

char *pszAsxSubSystemErrorString;
int nLen1,nLen2;

ASX_Error_GetLast( hObj, (ASX_ERROR*)&nError, &asxSubSystemErrorCode);
if(!nError)
    return 0;
ASX_Error_GetLastString( hObj, 0,0,&nLen1,0,0,&nLen2);
pszAsxErrorString = (char *)malloc(nLen1);
pszAsxSubSystemErrorString = (char *)malloc(nLen2);
ASX_Error_GetLastString( hObj, pszAsxErrorString,nLen1,&nLen1,pszAsxSubSystem
    ErrorString,nLen2,&nLen2);
printf("Error: %#d, %s - Subsystem Error: %#d, %s \n",
    nError,
    pszAsxErrorString,
    asxSubSystemErrorCode,
    pszAsxSubSystemErrorString );
printf("When called from source %s line %d\n",__FILE__,__LINE__);

printf("Press ENTER to exit\n");
getchar();
free(pszAsxErrorString);
free(pszAsxSubSystemErrorString);
ASX_System_Delete(hSystem);
exit(1);
return 1;
}

```

10.11 system/main.c

This is an example of how to use the ASX System functions.

```

/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/system/main
.c,v 1.5 2004/08/11 14:13:46 as-age Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"

int main(int argc, char* argv[])
{
    ASX_HANDLE system;
    ASX_ERROR asxError;
    char *pszAsxSubsysVersion;
    char *pszAsxVersion;
    int nAdapters=0;
    char *pszSystem;
    int nLen,nLen1,nLen2;

    printf("AudioScience ASX - System Example\n");

    asxError = ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&system);

    if(asxError)
    {
        int asxSubSystemErrorCode=0;
        char *pszAsxErrorString;
        char *pszAsxSubSystemErrorString;
    }
}

```

```

    ASX_Error_GetLast( system, &asxError, &asxSubSystemErrorCode);

    ASX_Error_GetLastString( system, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( system, pszAsxErrorString, nLen1, &nLen1, pszAsx
SubSystemErrorString, nLen2, &nLen2);
    printf("Error: #%d, %s - Subsystem Error: #%ld, %s \n", asxError, pszAsx
ErrorString,
    asxSubSystemErrorCode, pszAsxSubSystemErrorString );
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    return(0);
}

asxError = ASX_System_GetVersion(system, 0,0,&nLen1, 0, 0, &nLen2);
pszAsxVersion = (char *)malloc(nLen1);
pszAsxSubsysVersion = (char *)malloc(nLen2);
asxError = ASX_System_GetVersion(system, pszAsxVersion, nLen1, &nLen1, pszAsx
SubsysVersion, nLen2, &nLen2);

asxError = ASX_System_GetName( system, 0,0,&nLen );
pszSystem = (char *)malloc(nLen);
asxError = ASX_System_GetName( system, pszSystem,nLen,&nLen );
printf("System Ver %s\nSubsystem=%s Ver %s\n", pszAsxVersion, pszSystem, pszA
sxSubsysVersion);

asxError = ASX_System_GetAdapterCount(system,&nAdapters);
printf("There are %d audio adapters in the system \n", nAdapters);

printf("Press ENTER to exit\n");
getchar();
ASX_System_Delete(system);
free(pszAsxVersion);
free(pszAsxSubsysVersion);
free(pszSystem);
return(0);
}

```

10.12 tuner/main.c

This is an example of how to use the ASX tuner control functions.

```

/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/tuner/main.
c,v 1.9 2008/06/09 18:41:41 as-age Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"
#include "asxstring.h"

int CheckError(ASX_HANDLE hObj, const int nLine, const int nExitOnError);
void PrintControlName(ASX_HANDLE hControl);
ASX_HANDLE hSystem;

int main(int argc, char* argv[])
{
    char *pszName;

```

```

ASX_HANDLE hAdapter;
ASX_HANDLE hMixer;
ASX_HANDLE hTuner;
ASX_HANDLE hPAD;
ASX_ERROR asxError;
int nAdapterToUse=0;
int i,nLen;
unsigned long lFreq;
float fMin,fMax,fStep,fGain;
float fLevel;
enum asxTUNERBAND eBand;
int nCount;
char szString[64];
int nRequiredStringSize;
unsigned int nStatusBits,nStatusMask;

// create the system
ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
CheckError(hSystem, __LINE__,1);

// get the adapter
asxError = ASX_System_GetAdapter(hSystem,nAdapterToUse,&hAdapter);
CheckError(hSystem, __LINE__,1);

ASX_Adapter_GetName(hAdapter,0,0,&nLen);
CheckError(hAdapter, __LINE__,1);
pszName = (char *)malloc(nLen);
ASX_Adapter_GetName(hAdapter,pszName,nLen,&nLen);
CheckError(hAdapter, __LINE__,1);
printf("Adapter [%d] is %s \n", nAdapterToUse,pszName);

// get the mixer handle
asxError = ASX_Adapter_GetMixer( hAdapter, &hMixer );
CheckError(hAdapter, __LINE__,1);

// get a tuner object (index 0)
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_TUNER_IN,0,
    0,0,
    asxCONTROL_TUNER,
    &hTuner);
CheckError(hMixer, __LINE__,1);
// print out some control details
PrintControlName(hTuner);

// get Tuner band
asxError=ASX_Tuner_GetBand( hTuner, &eBand );
CheckError(hTuner, __LINE__,1);
if(!asxError)
{
    ASXSTRING_EnumToString(eBand, szString, 64, &nRequiredStringSize);
    printf("Tuner band is currently %s\n",szString);
}

// List all the bands and print them out
asxError=ASX_Tuner_EnumerateBand( hTuner, 0, &eBand, &nCount);

printf("Total tuner bands available is %d\n",nCount);
for(i=0;i<nCount;i++)
{

```

```

asxError=ASX_Tuner_EnumerateBand( hTuner, i, &eBand, &nCount);
CheckError(hTuner, __LINE__,1);

if(!asxError)
{
    ASXSTRING_EnumToString(eBand, szString, 64, &nRequiredStringSize);
    printf("Tuner band [%d] is %s\n",i,szString);
}
}

// setting Tuner to band FM
asxError=ASX_Tuner_SetBand( hTuner, asxTUNERBAND_FM );
CheckError(hTuner, __LINE__,1);

asxError = ASX_Tuner_GetFrequency( hTuner, &lFreq);
CheckError(hTuner, __LINE__,1);
asxError = ASX_Tuner_SetFrequency( hTuner, lFreq);
CheckError(hTuner, __LINE__,1);

asxError = ASX_Tuner_GetGainRange( hTuner, &fMin,&fMax,&fStep);
CheckError(hTuner, __LINE__,1);
if(!asxError)
{
    printf("Gain range, min=%f dB, Max = %f dB, Step size is %f dB\n",
        fMin,
        fMax,
        fStep);

    asxError = ASX_Tuner_GetGain( hTuner, &fGain);
    CheckError(hTuner, __LINE__, 0);
    asxError = ASX_Tuner_SetGain( hTuner, fGain);
    CheckError(hTuner, __LINE__, 0);
}

asxError = ASX_Tuner_GetRFLevel( hTuner, &fLevel);
CheckError(hTuner, __LINE__, 0);
printf("RF level is %f \n",fGain);

// get the status (not all tuners have status field) ?
asxError = ASX_Tuner_GetStatus( hTuner, &nStatusMask, &nStatusBits);
CheckError(hTuner, __LINE__, 0);
if(!asxError)
    printf("Tuner status mask 0x%08X, vale 0x%08X\n",nStatusMask,nStatusBits)
;

// read RDS fields

// get a PAD/RDS object (index 0)
asxError = ASX_Mixer_GetControlByNodeTypeAndIndex(
    hMixer,
    asxNODE_TUNER_IN,0,
    0,0,
    asxCONTROL_PAD,
    &hPAD);
CheckError(hMixer, __LINE__,0);
if(!asxError)
{
    char szBuffer[ASX_LONGLONG_STRING];
    int n;

```

```

    // print out some control details
    PrintControlName(hPAD);

    // channel name
    asxError = ASX_PAD_GetChannelName(hPAD, szBuffer, sizeof(szBuffer));
    CheckError(hMixer, __LINE__, 0);
    if(asxError==asxERROR_NO_ERROR)
        printf("PAD (ChannelName) : %s\n", szBuffer);

    // RDS PI
    asxError = ASX_PAD_GetRdsPI(hPAD, &n);
    CheckError(hMixer, __LINE__, 0);
    if(asxError==asxERROR_NO_ERROR)
        printf("PAD (RDS PI) : %d\n", n);

    // RDS PTY
    asxError = ASX_PAD_GetProgramType(hPAD, &n);
    CheckError(hMixer, __LINE__, 0);
    if(asxError==asxERROR_NO_ERROR)
    {
        asxError=ASX_PAD_GetProgramTypeString(
            hPAD,
            asxTUNER_RDS_TYPE_RDS,
            n,
            szBuffer,
            sizeof(szBuffer));

        printf("PAD (Program Type) : [%d] %s\n", n, szBuffer);
    }

    // artist
    asxError = ASX_PAD_GetArtist(hPAD, szBuffer, sizeof(szBuffer));
    CheckError(hMixer, __LINE__, 0);
    if(asxError==asxERROR_NO_ERROR)
        printf("PAD (Artist) : %s\n", szBuffer);

    // title
    asxError = ASX_PAD_GetTitle(hPAD, szBuffer, sizeof(szBuffer));
    CheckError(hMixer, __LINE__, 0);
    if(asxError==asxERROR_NO_ERROR)
        printf("PAD (Title) : %s\n", szBuffer);

    // comment
    asxError = ASX_PAD_GetComment(hPAD, szBuffer, sizeof(szBuffer));
    CheckError(hMixer, __LINE__, 0);
    if(asxError==asxERROR_NO_ERROR)
        printf("PAD (Comment) : %s\n", szBuffer);
}

printf("DONE.\n");
printf("Press ENTER to exit\n");
getchar();
ASX_System_Delete(hSystem);
return 0;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;

```

```

    int nLen;
    enum asxCONTROL eControl;

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl, 0, 0, &nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eControl, pszName, nLen, &nLen);
    printf("Control : %s\n", pszName);

    free(pszName);
}

int CheckError(ASX_HANDLE hObj, const int nLine, const int nExitOnError)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1, nLen2;

    ASX_Error_GetLast( hObj, &nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString, nLen1, &nLen1, pszAsxSubSystem
        ErrorString, nLen2, &nLen2);
    printf("Error: #%d, %s - Subsystem Error: #%ld, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
    printf("When called from source %s line %d\n", __FILE__, nLine);
    if(nExitOnError)
    {
        printf("Press ENTER to exit\n");
        getchar();
        free(pszAsxErrorString);
        free(pszAsxSubSystemErrorString);
        ASX_System_Delete(hSystem);
        exit(1);
    }
    return 1;
}

```

10.13 volume/main.c

This is an example of how to use the ASX Volume functions.

```

/* $Header: /home/eliot/asi/repo/cvsrepo/Repository/apps/asx/examples/volume/main
.c,v 1.2 2010/06/22 18:15:12 as-age Exp $ */
#include "stdio.h"
#include "stdlib.h"
#include "asx.h"
#include "asxstring.h"

ASX_HANDLE hSystem=0;

```

```

int CheckError(ASX_HANDLE hObj, const int nLine);
void PrintControlName(ASX_HANDLE hControl);

int main(int argc, char* argv[])
{
    ASX_HANDLE hAdapter;
    ASX_HANDLE hMixer;
    ASX_HANDLE hVolume;
    int nAdapterToUse=0;
    char szName[ASX_SHORT_STRING];
    int nLength;
    int nVol;
    float fGain[2];
    float fMin,fMax,fStep;
    int nChannels;

    // create the system
    ASX_System_Create(ASX_SYSTEM_TYPE_HPI,&hSystem);
    CheckError( hSystem, __LINE__);

    // get the adapter
    ASX_System_GetAdapter(hSystem,nAdapterToUse,&hAdapter);
    CheckError( hSystem, __LINE__);

    // get the adapter name
    ASX_Adapter_GetName(hAdapter,szName,sizeof(szName),&nLength);
    CheckError( hAdapter, __LINE__);
    printf("Adapter [%d] is %s \n", nAdapterToUse,szName);

    // get the mixer handle
    ASX_Adapter_GetMixer( hAdapter, &hMixer );
    CheckError( hAdapter, __LINE__);

    // get a volume object
    ASX_Mixer_GetControlByNodeTypeAndIndex(
        hMixer,
        asxNODE_PLAYER,0,    // play object, index 0
        asxNODE_LINE_OUT,0,  // line out object, index 0
        asxCONTROL_VOLUME,
        &hVolume);
    CheckError( hMixer, __LINE__);
    // print out control name
    PrintControlName(hVolume);

    ASX_Volume_GetRange( hVolume, &fMin, &fMax, &fStep);
    CheckError( hVolume, __LINE__);
    printf("Volume max %f, min %f, step %f\n",fMax,fMin,fStep);

    printf("Enter a volume to apply. 0 is fullscale, -100 is off.\n>");
    scanf("%d",&nVol);
    printf("Volume set.\n");
    fGain[0] = (float)nVol; // index 0 is the left channel
    fGain[1] = (float)nVol; // index 1 is the right channel

    ASX_Volume_GetChannels( hVolume, &nChannels);
    CheckError( hVolume, __LINE__);

    ASX_Volume_SetGain( hVolume, fGain, nChannels);
    CheckError( hVolume, __LINE__);

    printf("Press ENTER to exit\n");
}

```

```

    getchar();
    ASX_System_Delete(hSystem);
    return 0;
}

void PrintControlName(ASX_HANDLE hControl)
{
    char *pszName;
    int nLen;
    enum asxCONTROL eControl;

    ASX_Control_GetType(hControl, &eControl);
    ASXSTRING_EnumToString(eControl, 0, 0, &nLen);
    pszName=(char *)malloc(nLen);
    ASXSTRING_EnumToString(eControl, pszName, nLen, &nLen);
    printf("Control : %s\n", pszName);

    free(pszName);
}

int CheckError(ASX_HANDLE hObj, const int nLine)
{
    int nError;
    int asxSubSystemErrorCode=0;
    char *pszAsxErrorString;
    char *pszAsxSubSystemErrorString;
    int nLen1, nLen2;

    ASX_Error_GetLast( hObj, (ASX_ERROR*)&nError, &asxSubSystemErrorCode);
    if(!nError)
        return 0;
    ASX_Error_GetLastString( hObj, 0, 0, &nLen1, 0, 0, &nLen2);
    pszAsxErrorString = (char *)malloc(nLen1);
    pszAsxSubSystemErrorString = (char *)malloc(nLen2);
    ASX_Error_GetLastString( hObj, pszAsxErrorString, nLen1, &nLen1, pszAsxSubSystem
        ErrorString, nLen2, &nLen2);
    printf("Error: #%d, %s - Subsystem Error: #%ld, %s \n",
        nError,
        pszAsxErrorString,
        asxSubSystemErrorCode,
        pszAsxSubSystemErrorString );
    printf("When called from source %s line %d\n", __FILE__, nLine);

    printf("Press ENTER to exit\n");
    getchar();
    free(pszAsxErrorString);
    free(pszAsxSubSystemErrorString);
    ASX_System_Delete(hSystem);
    exit(1);
    return 1;
}

```

Index

- [_RPT0](#)
[asx.h, 202](#)
 - [_RPT1](#)
[asx.h, 202](#)
- [Adapter](#)
 - [ASX_Adapter_CheckSubSystems, 35](#)
 - [ASX_Adapter_EnumerateMode, 35](#)
 - [ASX_Adapter_EnumerateProperty, 35](#)
 - [ASX_Adapter_GetAVDECCController, 36](#)
 - [ASX_Adapter_GetAVDECCSystem, 36](#)
 - [ASX_Adapter_GetDspUtilization, 36](#)
 - [ASX_Adapter_GetFirmwareRevision, 37](#)
 - [ASX_Adapter_GetHardwareRevision, 37](#)
 - [ASX_Adapter_GetIndex, 38](#)
 - [ASX_Adapter_GetIpAddress, 38](#)
 - [ASX_Adapter_GetMacAddress, 39](#)
 - [ASX_Adapter_GetMixer, 39](#)
 - [ASX_Adapter_GetMode, 40](#)
 - [ASX_Adapter_GetName, 40](#)
 - [ASX_Adapter_GetNvMemSizeInBytes, 41](#)
 - [ASX_Adapter_GetSerialNumber, 41](#)
 - [ASX_Adapter_ReadNvMem, 42](#)
 - [ASX_Adapter_ReadProperty, 42](#)
 - [ASX_Adapter_SetMode, 43](#)
 - [ASX_Adapter_WriteNvMem, 43](#)
 - [ASX_Adapter_WriteProperty, 43](#)
- [Adapter functions, 33](#)
- [addr_end](#)
 - [asxCobranetIpAutoassignParameters, 163](#)
- [addr_start](#)
 - [asxCobranetIpAutoassignParameters, 163](#)
- [AESEBU receiver control functions, 110](#)
- [AESEBU transmitter control functions, 112](#)
- [AESEBU_RECEIVER_Aes3Rx](#)
 - [ASX_AESEBUReceiver_EnumerateFormat, 110](#)
 - [ASX_AESEBUReceiver_GetErrorStatus, 111](#)
 - [ASX_AESEBUReceiver_GetFormat, 111](#)
 - [ASX_AESEBUReceiver_GetSampleRate, 111](#)
 - [ASX_AESEBUReceiver_SetFormat, 112](#)
- [AESEBU_TRANSMITTER_Aes3Tx](#)
 - [ASX_AESEBUTransmitter_EnumerateFormat, 113](#)
 - [ASX_AESEBUTransmitter_GetFormat, 113](#)
 - [ASX_AESEBUTransmitter_SetFormat, 113](#)
- [ARRAY_SIZE](#)
 - [asx.h, 202](#)
- [asx.h, 167](#)
 - [_RPT0, 202](#)
 - [_RPT1, 202](#)
 - [ARRAY_SIZE, 202](#)
 - [ASX32_API, 202](#)
 - [ASX_ERROR, 202](#)
 - [ASX_ERROR_CALLBACK, 202](#)
 - [ASX_HANDLE, 203](#)
 - [ASX_LONG_STRING, 202](#)
 - [ASX_LONGLONG_STRING, 202](#)
 - [ASX_NODE, 203](#)
 - [ASX_PLAYER_CALLBACK, 203](#)
 - [ASX_SHORT_STRING, 202](#)
 - [ASX_TIME, 203](#)
 - [asxADAPTER_PROPERTY_ERRATA_1, 203](#)
 - [asxADAPTER_PROPERTY_FIRMWARE_ID, 203](#)
 - [asxADAPTER_PROPERTY_SXX2_SETTING, 203](#)

- asxADAPTER_PROPERTY_SUPPORT_-SSX2, [203](#)
- asxADAPTER_PROPERTY_SUPPORTS_-FW_UPDATE, [203](#)
- asxADAPTER_PROPERTY_SUPPORTS_-SNMP, [203](#)
- asxADAPTER_PROPERTY_SYNC_-HEADER_CONNECTIONS, [203](#)
- asxADAPTER_PROPERTY, [203](#)
- asxADAPTERMODE, [203](#)
- asxADAPTERMODE_12_PLAY, [204](#)
- asxADAPTERMODE_16_PLAY, [204](#)
- asxADAPTERMODE_1_PLAY, [204](#)
- asxADAPTERMODE_24_PLAY, [204](#)
- asxADAPTERMODE_32_PLAY, [204](#)
- asxADAPTERMODE_4_PLAY, [204](#)
- asxADAPTERMODE_6_PLAY, [204](#)
- asxADAPTERMODE_8_PLAY, [204](#)
- asxADAPTERMODE_9_PLAY, [204](#)
- asxADAPTERMODE_ILLEGAL, [204](#)
- asxADAPTERMODE_LOW_LATENCY, [204](#)
- asxADAPTERMODE_MODE_1, [204](#)
- asxADAPTERMODE_MODE_2, [204](#)
- asxADAPTERMODE_MODE_3, [204](#)
- asxADAPTERMODE_MONO, [204](#)
- asxADAPTERMODE_MULTICHANNEL, [204](#)
- asxADPROPENUM_MODE_PROPERTIES, [204](#)
- asxADPROPENUM_MODE_SETTINGS, [204](#)
- asxADPROPENUM_SX2_OFF, [204](#)
- asxADPROPENUM_SX2_ON, [204](#)
- asxADPROPENUM_MODE, [204](#)
- asxADPROPENUM_SX2, [204](#)
- asxAESEBU_ERROR, [205](#)
- asxAESEBU_ERROR_BIPHASE_VIOLATION, [205](#)
- asxAESEBU_ERROR_CHANNELSTATUS_CRC, [205](#)
- asxAESEBU_ERROR_NOT_LOCKED, [205](#)
- asxAESEBU_ERROR_PARITY_ERROR, [205](#)
- asxAESEBU_ERROR_POOR_QUALITY, [205](#)
- asxAESEBU_ERROR_VALIDITY, [205](#)
- asxAESEBU_FORMAT_AESEBU, [205](#)
- asxAESEBU_FORMAT_SPDIF, [205](#)
- asxAESEBU_FORMAT_UNDEFINED, [205](#)
- asxAESEBU_FORMAT, [204](#)
- asxAESEBU_STATUS, [205](#)
- asxAUDIO_FORMAT_DOLBY_AC2, [205](#)
- asxAUDIO_FORMAT_MPEG_AACPLUS, [205](#)
- asxAUDIO_FORMAT_MPEG_L2, [205](#)
- asxAUDIO_FORMAT_MPEG_L3, [205](#)
- asxAUDIO_FORMAT_NONE, [205](#)
- asxAUDIO_FORMAT_PCM16, [205](#)
- asxAUDIO_FORMAT_PCM20, [205](#)
- asxAUDIO_FORMAT_PCM24, [205](#)
- asxAUDIO_FORMAT_PCM32, [205](#)
- asxAUDIO_FORMAT_PCM32_FLOAT, [205](#)
- asxAUDIO_FORMAT_PCM8, [205](#)
- asxAUDIO_FORMAT, [205](#)
- asxCHANNELMODE, [205](#)
- asxCHANNELMODE_ILLEGAL, [206](#)
- asxCHANNELMODE_LEFTTOSTEREO, [206](#)
- asxCHANNELMODE_NORMAL, [206](#)
- asxCHANNELMODE_RIGHTTOSTEREO, [206](#)
- asxCHANNELMODE_STEREOTOLEFT, [206](#)
- asxCHANNELMODE_STEREOTORIGHT, [206](#)
- asxCHANNELMODE_SWAP, [206](#)
- asxCOBANET_IFSTATUS_ACTIVE_-CONNECTION, [206](#)
- asxCOBANET_IFSTATUS_FULL_-DUPLEX, [206](#)
- asxCOBANET_IFSTATUS_LINK_-ESTABLISHED, [206](#)
- asxCOBANET_LATENCY_133ms, [206](#)
- asxCOBANET_LATENCY_266ms, [206](#)
- asxCOBANET_LATENCY_533ms, [206](#)
- asxCOBANET_MODE_NETWORK, [207](#)
- asxCOBANET_MODE_TETHERED, [207](#)
- asxCOBANET_IFSTATUS, [206](#)
- asxCOBANET_LATENCY, [206](#)
- asxCOBANET_MODE, [206](#)

- asxCOMPANDER_INDEX_COMPANDER, 207
- asxCOMPANDER_INDEX_NOISEGATE, 207
- asxCOMPANDER_INDEX, 207
- asxCONTROL, 207
- asxCONTROL_AES18_BLOCK_GENERATOR, 207
- asxCONTROL_AES18_RECEIVER, 207
- asxCONTROL_AES18_TRANSMITTER, 207
- asxCONTROL_AESEBU_RECEIVER, 207
- asxCONTROL_AESEBU_TRANSMITTER, 207
- asxCONTROL_BIT_STREAM, 208
- asxCONTROL_BLOCK, 208
- asxCONTROL_CHANNEL_MODE, 208
- asxCONTROL_COBRANET, 208
- asxCONTROL_COBRANET_RECEIVER, 208
- asxCONTROL_COBRANET_TRANSMITTER, 208
- asxCONTROL_COMPANDER, 208
- asxCONTROL_CONNECTION, 207
- asxCONTROL_GENERIC, 208
- asxCONTROL_GPIO, 208
- asxCONTROL_INVALID, 207
- asxCONTROL_LAST_CONTROL, 208
- asxCONTROL_LEVEL, 207
- asxCONTROL_METER, 207
- asxCONTROL_MICROPHONE, 208
- asxCONTROL_MULTIPLEXER, 207
- asxCONTROL_MUTE, 207
- asxCONTROL_PAD, 208
- asxCONTROL_PARAMETRIC_EQ, 208
- asxCONTROL_PLAYER, 208
- asxCONTROL_RDS, 207
- asxCONTROL_RECORDER, 208
- asxCONTROL_RESERVED_525, 208
- asxCONTROL_RESERVED_526, 208
- asxCONTROL_RESERVED_527, 208
- asxCONTROL_RESERVED_528, 208
- asxCONTROL_SAMPLE_CLOCK, 208
- asxCONTROL_SILENCEDETECTOR, 208
- asxCONTROL_SRC, 208
- asxCONTROL_TONEDETECTOR, 208
- asxCONTROL_TUNER, 207
- asxCONTROL_VOLUME, 207
- asxCONTROL_VOX, 207
- asxEQBANDTYPE, 208
- asxEQBANDTYPE_BANDPASS, 209
- asxEQBANDTYPE_BANDSTOP, 209
- asxEQBANDTYPE_BYPASS, 208
- asxEQBANDTYPE_EQUALIZER, 209
- asxEQBANDTYPE_HIGHPASS, 209
- asxEQBANDTYPE_HIGHSHELF, 208
- asxEQBANDTYPE_LOWPASS, 209
- asxEQBANDTYPE_LOWSHELF, 208
- asxERROR, 209
- asxERROR_AES18, 209
- asxERROR_ALREADY_OPEN, 209
- asxERROR_ASXObject, 209
- asxERROR_BUFFER_TOO_SMALL, 209
- asxERROR_COBRANET_NODE_FOUND, 210
- asxERROR_COBRANET_NODE_NOT_FOUND, 210
- asxERROR_COBRANET_NODE_UNREACHABLE, 210
- asxERROR_COMMUNICATING_WITH_DEVICE, 209
- asxERROR_CONTROL_NOT_READY, 210
- asxERROR_DEPRECATED, 210
- asxERROR_DISCO_DLL_NOT_FOUND, 210
- asxERROR_DUPLICATE_ADAPTER_INDEX, 210
- asxERROR_ENUMERATE_INDEX_OUT_OF_RANGE, 209
- asxERROR_FILE_OPEN_FAILED, 211
- asxERROR_HOST_NOT_FOUND, 210
- asxERROR_INDEX_OUT_OF_RANGE, 209
- asxERROR_INTERNAL_BUFFERING_ERROR, 209
- asxERROR_INVALID_CONTROL, 210
- asxERROR_INVALID_CONTROL_ATTRIBUTE, 210
- asxERROR_INVALID_CONTROL_NOT_FOUND, 210
- asxERROR_INVALID_CONTROL_OPERATION, 210

- asxERROR_INVALID_CONTROL_-
VALUE, 210
- asxERROR_INVALID_FORMAT, 209
- asxERROR_INVALID_NUMBER_-
OF_CHANNELS, 210
- asxERROR_INVALID_OPERATION,
209
- asxERROR_IP_ASSIGNED, 210
- asxERROR_IP_AUTOASSIGN_DISABLED,
210
- asxERROR_IP_CHANGED, 210
- asxERROR_MIXER_SAVECONTROLSTATE,
211
- asxERROR_NO_ERROR, 209
- asxERROR_NO_IP_ADDRESSES_-
AVAILABLE, 210
- asxERROR_NOT_OPEN, 209
- asxERROR_OUTOFMEMORY, 210
- asxERROR_PCAP_ERROR, 210
- asxERROR_PLAYER_FILEOPENERERROR,
211
- asxERROR_PLAYER_FILEREADERERROR,
211
- asxERROR_PLAYER_INTERNAL_-
STATE_FAILURE, 211
- asxERROR_PLAYER_INVALIDFILEFORMAT,
211
- asxERROR_PLAYER_NOFILE, 211
- asxERROR_PLAYER_OUT_OF_SEQUENCE_-
CALL, 211
- asxERROR_PLAYER_TIME_OUT, 211
- asxERROR_PLAYER_TWAV, 211
- asxERROR_PLAYER_UNSUPPORTEDFORMAT,
211
- asxERROR_RECORDER_FILECREATEERROR,
211
- asxERROR_RECORDER_FILEWRITEERROR,
211
- asxERROR_RECORDER_FORMATMISMATCH,
211
- asxERROR_RECORDER_INTERNAL_-
STATE_FAILURE, 211
- asxERROR_RECORDER_INVALIDFILENAME,
211
- asxERROR_RECORDER_OUT_OF_-
SEQUENCE_CALL, 211
- asxERROR_RECORDER_TIME_OUT,
211
- asxERROR_RECORDER_TWAV, 211
- asxERROR_STARTING_DEVICE, 209
- asxERROR_TOO_MANY_CLIENTS,
210
- asxERROR_UNIMPLEMENTED, 209
- asxERROR_UNKNOWN, 211
- asxERROR_UNSUPPORTED_CONTROL_-
ATTRIBUTE, 210
- asxFILE_FORMAT_RAW, 212
- asxFILE_FORMAT_WAV, 212
- asxFILE_MODE_APPEND, 212
- asxFILE_MODE_CREATE, 212
- asxFILE_FORMAT, 211
- asxFILE_MODE, 212
- asxHANDLE_ADAPTER, 212
- asxHANDLE_CONTROL, 212
- asxHANDLE_INVALID, 212
- asxHANDLE_LAST, 212
- asxHANDLE_MIXER, 212
- asxHANDLE_NODE, 212
- asxHANDLE_SYSTEM, 212
- asxHANDLE_TYPE, 212
- asxMETER_PEAK, 212
- asxMETER_RMS, 212
- asxMETER_TYPE, 212
- asxMSG_LOGGING_DEBUG, 213
- asxMSG_LOGGING_DISABLE, 213
- asxMSG_LOGGING_ERROR, 213
- asxMSG_LOGGING_INFO, 213
- asxMSG_LOGGING_NOTICE, 213
- asxMSG_LOGGING_VERBOSE, 213
- asxMSG_LOGGING_WARNING, 213
- asxMSG_LOGGING, 212
- asxNODE, 213
- asxNODE_ADAPTER, 213
- asxNODE_AESEBU_IN, 213
- asxNODE_AESEBU_OUT, 214
- asxNODE_ANALOG_IN, 214
- asxNODE_ANALOG_OUT, 214
- asxNODE_AVB_IN_AUDIO, 214
- asxNODE_AVB_IN_STREAM, 214
- asxNODE_AVB_OUT_AUDIO, 214
- asxNODE_AVB_OUT_STREAM, 214
- asxNODE_BITSTREAM_IN, 213
- asxNODE_BLULINK_IN, 214
- asxNODE_BLULINK_OUT, 214
- asxNODE_CLOCK_SOURCE_IN, 213
- asxNODE_COBRANET_IN, 213
- asxNODE_COBRANET_OUT, 214
- asxNODE_COBRANET_RECEIVER,
213

- asxNODE_COBRANET_TRANSMITTER, 214
- asxNODE_FIRST_DEST_NODE, 214
- asxNODE_INTERNAL_IN, 214
- asxNODE_INTERNAL_OUT, 214
- asxNODE_INVALID, 213
- asxNODE_LAST_DEST_NODE, 214
- asxNODE_LAST_SOURCE_NODE, 214
- asxNODE_LINE_IN, 213
- asxNODE_LINE_OUT, 214
- asxNODE_MICROPHONE_IN, 213
- asxNODE_NONE, 213
- asxNODE_PLAYER, 213
- asxNODE_RADIO_FREQ_IN, 213
- asxNODE_RADIO_FREQ_OUT, 214
- asxNODE_RECORDER, 214
- asxNODE_RTP_DESTINATION_IN, 214
- asxNODE_RTP_SOURCE_OUT, 214
- asxNODE_SDI_IN, 214
- asxNODE_SDI_OUT, 214
- asxNODE_SPEAKER_OUT, 214
- asxNODE_TUNER_IN, 213
- asxPARAM_FLAG_READABLE, 221
- asxPARAM_FLAG_VOLATILE, 221
- asxPARAM_FLAG_WRITEABLE, 221
- asxPARAM_RANGE_ENUMERATED, 222
- asxPARAM_RANGE_ENUMERATED_FLOAT, 222
- asxPARAM_RANGE_ENUMERATED_INTEGER, 222
- asxPARAM_RANGE_NONE, 222
- asxPARAM_RANGE_NUMBER_OF_BITS, 222
- asxPARAM_RANGE_STEPPED_FLOAT, 222
- asxPARAM_RANGE_STEPPED_INTEGER, 222
- asxPARAM_RANGE_STRING_LENGTH, 222
- asxPARAM_TYPE_BOOLEAN, 221
- asxPARAM_TYPE_DOUBLE, 221
- asxPARAM_TYPE_FLOAT, 221
- asxPARAM_TYPE_INTEGER, 221
- asxPARAM_TYPE_IP4_ADDRESS, 221
- asxPARAM_TYPE_IP6_ADDRESS, 221
- asxPARAM_TYPE_MAC_ADDRESS, 221
- asxPARAM_TYPE_NONE, 221
- asxPARAM_TYPE_STRING, 221
- asxPLAYER_DESTROY, 215
- asxPLAYER_DONE, 215
- asxPLAYER_FILE_COMPLETE, 214
- asxPLAYER_FILE_START, 214
- asxPLAYER_FILELIST_COMPLETE, 214
- asxPLAYER_INIT, 215
- asxPLAYER_OPEN, 215
- asxPLAYER_PAUSED, 215
- asxPLAYER_PREFILL, 215
- asxPLAYER_RUNNING, 215
- asxPLAYER_FLAGS, 214
- asxPLAYER_STATE, 214
- asxRECORD_MODE_DONT_CARE, 215
- asxRECORD_MODE_DUAL_MONO, 215
- asxRECORD_MODE_JOINT_STEREO, 215
- asxRECORD_MODE_MONO, 215
- asxRECORD_MODE_STEREO, 215
- asxRECORD_MODE, 215
- asxRECORDER_DESTROY, 215
- asxRECORDER_DONE, 215
- asxRECORDER_INIT, 215
- asxRECORDER_OPEN, 215
- asxRECORDER_PAUSED, 215
- asxRECORDER_RUNNING, 215
- asxRECORDER_STATE, 215
- asxSAMPLE_CLOCK_SOURCE_ADAPTER, 216
- asxSAMPLE_CLOCK_SOURCE_AESEBUAUTO, 216
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT1, 216
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT10, 216
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT11, 216
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT12, 216
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT13, 216
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT14, 217

asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT1	asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT9,
217	216
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT10	asxSAMPLE_CLOCK_SOURCE_AESEBUSYNC,
217	216
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT11	asxSAMPLE_CLOCK_SOURCE_BLULINK,
217	218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT12	asxSAMPLE_CLOCK_SOURCE_LIVEWIRE,
217	218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT13	asxSAMPLE_CLOCK_SOURCE_LOCAL,
217	217
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT14	asxSAMPLE_CLOCK_SOURCE_NETWORK,
216	216
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT15	asxSAMPLE_CLOCK_SOURCE_PREV_-
217	MODULE, 217
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT16	asxSAMPLE_CLOCK_SOURCE_SMPTE,
217	216
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT17	asxSAMPLE_CLOCK_SOURCE_UNDEFINED,
217	217
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT18	asxSAMPLE_CLOCK_SOURCE_WORD,
217	216
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT19	asxSAMPLE_CLOCK_SOURCE_WORD_-
217	HEADER, 216
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT20	asxSAMPLE_RATE_11025, 218
217	asxSAMPLE_RATE_12000, 218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT21	asxSAMPLE_RATE_16000, 218
217	asxSAMPLE_RATE_176400, 218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT22	asxSAMPLE_RATE_192000, 218
217	asxSAMPLE_RATE_22050, 218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT23	asxSAMPLE_RATE_24000, 218
217	asxSAMPLE_RATE_32000, 218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT24	asxSAMPLE_RATE_44100, 218
217	asxSAMPLE_RATE_48000, 218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT25	asxSAMPLE_RATE_64000, 218
217	asxSAMPLE_RATE_8000, 218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT26	asxSAMPLE_RATE_88200, 218
217	asxSAMPLE_RATE_96000, 218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT27	asxSAMPLE_RATE_UNDEFINED, 218
217	asxSAMPLE_CLOCK_SOURCE, 215
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT28	asxSAMPLE_RATE, 218
217	asxTIMESCALE, 218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT29	asxTIMESCALE_BYTES, 218
216	asxTIMESCALE_BYTES_REMAINING,
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT30	asxTIME,
217	218
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT31	asxTIMESCALE_INVALID, 218
217	asxTIMESCALE_MILLISECONDS,
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT32	asxTIMESCALE_MILLISECONDS_-
217	REMA
asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT33	asxTIMESCALE_SAMPLES, 218
216	

- asxTIMESCALE_SAMPLES_REMAINING, 219
- asxTUNER_RDS_TYPE_RBDS, 219
- asxTUNER_RDS_TYPE_RDS, 219
- asxTUNER_STATUS_DIGITAL, 219
- asxTUNER_STATUS_FIRMWARE_LOADING, 219
- asxTUNER_STATUS_FM_STEREO, 219
- asxTUNER_STATUS_MULTIPROGRAM, 219
- asxTUNER_STATUS_PLL_LOCKED, 219
- asxTUNER_STATUS_VIDEO_COLOR_PRESENT, 219
- asxTUNER_STATUS_VIDEO_HORZ_SYNC_MISSING, 219
- asxTUNER_STATUS_VIDEO_IS_60HZ, 219
- asxTUNER_STATUS_VIDEO_VALID, 219
- asxTUNER_RDS_TYPE, 219
- asxTUNER_STATUS, 219
- asxTUNERBAND, 219
- asxTUNERBAND_AM, 219
- asxTUNERBAND_AUX, 219
- asxTUNERBAND_DAB, 220
- asxTUNERBAND_FM, 219
- asxTUNERBAND_FM_STEREO, 219
- asxTUNERBAND_TV, 219
- asxTUNERBAND_TV_PAL_BG, 220
- asxTUNERBAND_TV_PAL_DK, 220
- asxTUNERBAND_TV_PAL_I, 220
- asxTUNERBAND_TV_SECAM_L, 220
- asxTUNERDEEMPHASIS, 220
- asxTUNERDEEMPHASIS_50, 220
- asxTUNERDEEMPHASIS_75, 220
- asxTUNERDEEMPHASIS_NONE, 220
- asxTUNERHDBLEND, 220
- asxTUNERHDBLEND_ANALOG, 220
- asxTUNERHDBLEND_AUTO, 220
- asxTUNERMODE, 220
- asxTUNERMODE_RSS, 220
- asxTUNERMODE_RSS_DISABLE, 220
- asxTUNERMODE_RSS_ENABLE, 220
- asxTUNERPROGRAM, 220
- asxTUNERPROGRAM_1, 220
- asxTUNERPROGRAM_2, 221
- asxTUNERPROGRAM_3, 221
- asxTUNERPROGRAM_4, 221
- asxTUNERPROGRAM_5, 221
- asxTUNERPROGRAM_6, 221
- asxTUNERPROGRAM_7, 221
- asxTUNERPROGRAM_8, 221
- asxTUNERPROGRAM_NONE, 220
- asxUCONTROL_PFLAGS, 221
- asxUCONTROL_PTYPE, 221
- asxUCONTROL_RTYPE, 221
- asxVOLUME_AUTOFADE_LINEAR, 222
- asxVOLUME_AUTOFADE_LOG, 222
- asxVOLUME_AUTOFADE, 222
- ASX32_API
- asx.h, 202
- asxstring.h, 223
- ASX_Adapter_CheckSubSystems
- Adapter, 35
- ASX_Adapter_EnumerateMode
- Adapter, 35
- ASX_Adapter_EnumerateProperty
- Adapter, 35
- ASX_Adapter_GetAVDECCController
- Adapter, 36
- ASX_Adapter_GetAVDECCSystem
- Adapter, 36
- ASX_Adapter_GetDspUtilization
- Adapter, 36
- ASX_Adapter_GetFirmwareRevision
- Adapter, 37
- ASX_Adapter_GetHardwareRevision
- Adapter, 37
- ASX_Adapter_GetIndex
- Adapter, 38
- ASX_Adapter_GetIpAddress
- Adapter, 38
- ASX_Adapter_GetMacAddress
- Adapter, 39
- ASX_Adapter_GetMixer
- Adapter, 39
- ASX_Adapter_GetMode
- Adapter, 40
- ASX_Adapter_GetName
- Adapter, 40
- ASX_Adapter_GetNvMemSizeInBytes
- Adapter, 41
- ASX_Adapter_GetSerialNumber
- Adapter, 41
- ASX_Adapter_ReadNvMem
- Adapter, 42

- ASX_Adapter_ReadProperty
 - Adapter, [42](#)
- ASX_Adapter_SetMode
 - Adapter, [43](#)
- ASX_Adapter_WriteNvMem
 - Adapter, [43](#)
- ASX_Adapter_WriteProperty
 - Adapter, [43](#)
- ASX_AESEBUReceiver_EnumerateFormat
 - AESEBU_RECEIVER_Aes3Rx, [110](#)
- ASX_AESEBUReceiver_GetErrorStatus
 - AESEBU_RECEIVER_Aes3Rx, [111](#)
- ASX_AESEBUReceiver_GetFormat
 - AESEBU_RECEIVER_Aes3Rx, [111](#)
- ASX_AESEBUReceiver_GetSampleRate
 - AESEBU_RECEIVER_Aes3Rx, [111](#)
- ASX_AESEBUReceiver_SetFormat
 - AESEBU_RECEIVER_Aes3Rx, [112](#)
- ASX_AESEBUTransmitter_EnumerateFormat
 - AESEBU_TRANSMITTER_Aes3Tx, [113](#)
- ASX_AESEBUTransmitter_GetFormat
 - AESEBU_TRANSMITTER_Aes3Tx, [113](#)
- ASX_AESEBUTransmitter_SetFormat
 - AESEBU_TRANSMITTER_Aes3Tx, [113](#)
- ASX_Block_GetInfo
 - BLOCK_Block, [157](#)
- ASX_Block_Parameter_Get
 - BLOCK_Block, [157](#)
- ASX_Block_Parameter_GetElementCount
 - BLOCK_Block, [158](#)
- ASX_Block_Parameter_GetEnumName
 - BLOCK_Block, [158](#)
- ASX_Block_Parameter_GetFlags
 - BLOCK_Block, [159](#)
- ASX_Block_Parameter_GetName
 - BLOCK_Block, [159](#)
- ASX_Block_Parameter_GetRange
 - BLOCK_Block, [159](#)
- ASX_Block_Parameter_GetType
 - BLOCK_Block, [160](#)
- ASX_Block_Parameter_GetUnits
 - BLOCK_Block, [160](#)
- ASX_Block_Parameter_Set
 - BLOCK_Block, [161](#)
- ASX_ChannelMode_Enumerate
 - CHANNEL_MODE_Mode, [85](#)
- ASX_ChannelMode_Get
 - CHANNEL_MODE_Mode, [85](#)
- ASX_ChannelMode_Set
 - CHANNEL_MODE_Mode, [85](#)
- ASX_Cobranet_EnumerateModes
 - Cobranet, [130](#)
- ASX_Cobranet_GetConductorPriority
 - Cobranet, [130](#)
- ASX_Cobranet_GetConductorStatus
 - Cobranet, [130](#)
- ASX_Cobranet_GetDescription
 - Cobranet, [131](#)
- ASX_Cobranet_GetErrorInfo
 - Cobranet, [131](#)
- ASX_Cobranet_GetFirmwareRevision
 - Cobranet, [131](#)
- ASX_Cobranet_GetIfStatus
 - Cobranet, [132](#)
- ASX_Cobranet_GetIPAddress
 - Cobranet, [132](#)
- ASX_Cobranet_GetLatencyAndSampleRate
 - Cobranet, [133](#)
- ASX_Cobranet_GetLocation
 - Cobranet, [133](#)
- ASX_Cobranet_GetMACAddress
 - Cobranet, [133](#)
- ASX_Cobranet_GetMode
 - Cobranet, [134](#)
- ASX_Cobranet_GetName
 - Cobranet, [134](#)
- ASX_Cobranet_GetPersistence
 - Cobranet, [134](#)
- ASX_Cobranet_GetSerialConfig
 - Cobranet, [135](#)
- ASX_Cobranet_GetSerialEnable
 - Cobranet, [135](#)
- ASX_Cobranet_GetStaticIPAddress
 - Cobranet, [135](#)
- ASX_Cobranet_SetConductorPriority
 - Cobranet, [136](#)
- ASX_Cobranet_SetIPAddress
 - Cobranet, [136](#)
- ASX_Cobranet_SetLatencyAndSampleRate
 - Cobranet, [136](#)
- ASX_Cobranet_SetLocation
 - Cobranet, [137](#)
- ASX_Cobranet_SetMode
 - Cobranet, [137](#)
- ASX_Cobranet_SetName
 - Cobranet, [137](#)
- ASX_Cobranet_SetPersistence
 - Cobranet, [137](#)

- Cobranet, [138](#)
- ASX_Cobranet_SetSerialConfig
 - Cobranet, [138](#)
- ASX_Cobranet_SetSerialEnable
 - Cobranet, [138](#)
- ASX_Cobranet_SetStaticIPAddress
 - Cobranet, [139](#)
- ASX_CobranetRx_GetBundle
 - COBRANET_RECEIVER_CobranetRx, [145](#)
- ASX_CobranetRx_GetChannelMap
 - COBRANET_RECEIVER_CobranetRx, [146](#)
- ASX_CobranetRx_GetMinimumDelay
 - COBRANET_RECEIVER_CobranetRx, [146](#)
- ASX_CobranetRx_GetSourceMAC
 - COBRANET_RECEIVER_CobranetRx, [146](#)
- ASX_CobranetRx_GetStatus
 - COBRANET_RECEIVER_CobranetRx, [147](#)
- ASX_CobranetRx_SetBundle
 - COBRANET_RECEIVER_CobranetRx, [148](#)
- ASX_CobranetRx_SetChannelMap
 - COBRANET_RECEIVER_CobranetRx, [148](#)
- ASX_CobranetRx_SetMinimumDelay
 - COBRANET_RECEIVER_CobranetRx, [149](#)
- ASX_CobranetRx_SetSourceMAC
 - COBRANET_RECEIVER_CobranetRx, [149](#)
- ASX_CobranetTx_GetBundle
 - COBRANET_TRANSMITTER_CobranetTx, [140](#)
- ASX_CobranetTx_GetChannelCount
 - COBRANET_TRANSMITTER_CobranetTx, [140](#)
- ASX_CobranetTx_GetChannelMap
 - COBRANET_TRANSMITTER_CobranetTx, [141](#)
- ASX_CobranetTx_GetFormat
 - COBRANET_TRANSMITTER_CobranetTx, [141](#)
- ASX_CobranetTx_GetStatus
 - COBRANET_TRANSMITTER_CobranetTx, [142](#)
- ASX_CobranetTx_GetUnicastMode
 - COBRANET_TRANSMITTER_CobranetTx, [142](#)
- ASX_CobranetTx_SetBundle
 - COBRANET_TRANSMITTER_CobranetTx, [142](#)
- ASX_CobranetTx_SetChannelCount
 - COBRANET_TRANSMITTER_CobranetTx, [143](#)
- ASX_CobranetTx_SetChannelMap
 - COBRANET_TRANSMITTER_CobranetTx, [143](#)
- ASX_CobranetTx_SetFormat
 - COBRANET_TRANSMITTER_CobranetTx, [143](#)
- ASX_CobranetTx_SetUnicastMode
 - COBRANET_TRANSMITTER_CobranetTx, [144](#)
- ASX_Compander_Get
 - Compander, [123](#)
- ASX_Compander_GetAttackTimeConstant
 - Compander, [123](#)
- ASX_Compander_GetDecayTimeConstant
 - Compander, [124](#)
- ASX_Compander_GetEnable
 - Compander, [124](#)
- ASX_Compander_GetMakeupGain
 - Compander, [124](#)
- ASX_Compander_GetRatio
 - Compander, [125](#)
- ASX_Compander_GetThreshold
 - Compander, [125](#)
- ASX_Compander_Set
 - Compander, [125](#)
- ASX_Compander_SetAttackTimeConstant
 - Compander, [126](#)
- ASX_Compander_SetDecayTimeConstant
 - Compander, [126](#)
- ASX_Compander_SetEnable
 - Compander, [127](#)
- ASX_Compander_SetMakeupGain
 - Compander, [127](#)
- ASX_Compander_SetRatio
 - Compander, [127](#)
- ASX_Compander_SetThreshold
 - Compander, [128](#)
- ASX_Control_GetDestinationNode
 - CONTROL_ControlBase, [55](#)
- ASX_Control_GetHpiControl
 - CONTROL_ControlBase, [55](#)
- ASX_Control_GetSourceNode
 - CONTROL_ControlBase, [55](#)

- CONTROL_ControlBase, 55
- ASX_Control_GetSubSystem
 - CONTROL_ControlBase, 56
- ASX_Control_GetType
 - CONTROL_ControlBase, 56
- ASX_EQ_GetBand
 - PARAMETRIC_EQ_ParametricEQ, 120
- ASX_EQ_GetInfo
 - PARAMETRIC_EQ_ParametricEQ, 120
- ASX_EQ_SetBand
 - PARAMETRIC_EQ_ParametricEQ, 121
- ASX_EQ_SetState
 - PARAMETRIC_EQ_ParametricEQ, 121
- ASX_ERROR
 - asx.h, 202
- ASX_ERROR_CALLBACK
 - asx.h, 202
- ASX_Error_Clear
 - ERROR_Base, 31
- ASX_Error_GetLast
 - ERROR_Base, 31
- ASX_Error_GetLastString
 - ERROR_Base, 32
- ASX_GetGenericControlName
 - GENERIC_GenericControl, 118
- ASX_GPIO_GetProperties
 - Gpio, 114
- ASX_GPIO_InputGet
 - Gpio, 115
- ASX_GPIO_OutputGet
 - Gpio, 115
- ASX_GPIO_OutputSet
 - Gpio, 116
- ASX_HANDLE
 - asx.h, 203
- ASX_Handle_GetType
 - Handle, 30
- ASX_Level_Get
 - Level, 81
- ASX_Level_GetRange
 - Level, 82
- ASX_Level_Set
 - Level, 82
- ASX_LONG_STRING
 - asx.h, 202
- ASX_LONGLONG_STRING
 - asx.h, 202
- ASX_Meter_GetBallistics
 - Meter, 74
- ASX_Meter_GetChannels
 - Meter, 75
- ASX_Meter_GetPeak
 - Meter, 75
- ASX_Meter_GetRMS
 - Meter, 76
- ASX_Meter_SetBallistics
 - Meter, 76
- ASX_Mic_GetPhantomPower
 - MICROPHONE_Mic, 119
- ASX_Mic_SetPhantomPower
 - MICROPHONE_Mic, 119
- ASX_Mixer_GetBlockControlByNodeTypeAndIndex
 - Mixer, 45
- ASX_Mixer_GetControl
 - Mixer, 45
- ASX_Mixer_GetControlByNode
 - Mixer, 46
- ASX_Mixer_GetControlByNodeTypeAndIndex
 - Mixer, 46
- ASX_Mixer_GetControlCount
 - Mixer, 47
- ASX_Mixer_GetDestinationNode
 - Mixer, 48
- ASX_Mixer_GetDestinationNodeCount
 - Mixer, 48
- ASX_Mixer_GetNodeByType
 - Mixer, 48
- ASX_Mixer_GetNodeIndex
 - Node, 52
- ASX_Mixer_GetNodeType
 - Node, 52
- ASX_Mixer_GetNodeTypeCount
 - Mixer, 49
- ASX_Mixer_GetSourceNode
 - Mixer, 49
- ASX_Mixer_GetSourceNodeCount
 - Mixer, 50
- ASX_Mixer_ResetControls
 - Mixer, 50
- ASX_Multiplexer_Enumerate
 - MULTIPLEXER_Mux, 83
- ASX_Multiplexer_Get
 - MULTIPLEXER_Mux, 83
- ASX_Multiplexer_Set
 - MULTIPLEXER_Mux, 84
- ASX_NODE
 - asx.h, 203
- ASX_Node_GetIndex
 - Node, 52
- ASX_Node_GetLocation

- Node, [52](#)
- ASX_Node_GetName
 - Node, [53](#)
- ASX_Node_GetSubSystem
 - Node, [53](#)
- ASX_Node_GetType
 - Node, [53](#)
- ASX_PAD_GetArtist
 - Pad, [101](#)
- ASX_PAD_GetChannelName
 - Pad, [102](#)
- ASX_PAD_GetComment
 - Pad, [102](#)
- ASX_PAD_GetProgramType
 - Pad, [103](#)
- ASX_PAD_GetProgramTypeString
 - Pad, [103](#)
- ASX_PAD_GetRdsPI
 - Pad, [104](#)
- ASX_PAD_GetTitle
 - Pad, [104](#)
- ASX_PLAYER_CALLBACK
 - asx.h, [203](#)
- ASX_Player_Close
 - Player, [59](#)
- ASX_Player_Format_GetDetails
 - Player, [59](#)
- ASX_Player_Format_GetString
 - Player, [60](#)
- ASX_Player_GetFilename
 - Player, [60](#)
- ASX_Player_GetLoopMode
 - Player, [61](#)
- ASX_Player_GetPosition
 - Player, [61](#)
- ASX_Player_GetState
 - Player, [61](#)
- ASX_Player_GetTimeScale
 - Player, [62](#)
- ASX_Player_Open
 - Player, [62](#)
- ASX_Player_OpenPlaylist
 - Player, [63](#)
- ASX_Player_Pause
 - Player, [64](#)
- ASX_Player_PlaylistStatus
 - Player, [64](#)
- ASX_Player_PlaylistWait
 - Player, [65](#)
- ASX_Player_PreLoad
 - Player, [65](#)
- ASX_Player_RegisterCallback
 - Player, [66](#)
- ASX_Player_SetLoopMode
 - Player, [66](#)
- ASX_Player_SetPosition
 - Player, [66](#)
- ASX_Player_SetTimeScale
 - Player, [67](#)
- ASX_Player_Start
 - Player, [67](#)
- ASX_Player_Stop
 - Player, [67](#)
- ASX_Player_Wait
 - Player, [68](#)
- ASX_Recorder_Close
 - Recorder, [70](#)
- ASX_Recorder_EnumerateFormat
 - Recorder, [70](#)
- ASX_Recorder_GetFilename
 - Recorder, [71](#)
- ASX_Recorder_GetPosition
 - Recorder, [71](#)
- ASX_Recorder_GetState
 - Recorder, [72](#)
- ASX_Recorder_Open
 - Recorder, [72](#)
- ASX_Recorder_Pause
 - Recorder, [73](#)
- ASX_Recorder_Start
 - Recorder, [73](#)
- ASX_Recorder_Stop
 - Recorder, [73](#)
- ASX_SampleClock_EnumerateClockSource
 - SAMPLE_CLOCK_SampleClock, [106](#)
- ASX_SampleClock_EnumerateLocalRate
 - SAMPLE_CLOCK_SampleClock, [106](#)
- ASX_SampleClock_EnumerateSampleRate
 - SAMPLE_CLOCK_SampleClock, [106](#)
- ASX_SampleClock_GetAutoSource
 - SAMPLE_CLOCK_SampleClock, [106](#)
- ASX_SampleClock_GetClockSource
 - SAMPLE_CLOCK_SampleClock, [107](#)
- ASX_SampleClock_GetLocalRate
 - SAMPLE_CLOCK_SampleClock, [107](#)
- ASX_SampleClock_GetLocalRateLock
 - SAMPLE_CLOCK_SampleClock, [107](#)
- ASX_SampleClock_GetSampleRate
 - SAMPLE_CLOCK_SampleClock, [108](#)
- ASX_SampleClock_SetAutoSource

- SAMPLE_CLOCK_SampleClock, 108
- ASX_SampleClock_SetClockSource System, 26
- SAMPLE_CLOCK_SampleClock, 108
- ASX_SampleClock_SetLocalRate System, 26
- SAMPLE_CLOCK_SampleClock, 109
- ASX_SampleClock_SetLocalRateLock System, 27
- SAMPLE_CLOCK_SampleClock, 109
- ASX_SampleClock_SetSampleRate System, 27
- SAMPLE_CLOCK_SampleClock, 110
- ASX_SHORT_STRING System, 28
- asx.h, 202
- ASX_System_SetCobranetAutoassignParms
- ASX_SilenceDetector_GetDelay System, 28
- SILENCEDETECTOR_SilenceDetector, 154
- ASX_System_SetHostNetworkInterface System, 29
- ASX_SilenceDetector_GetEnable ASX_System_SetMessageLogging System, 29
- SILENCEDETECTOR_SilenceDetector, 154
- ASX_System_SupportsSubSystem System, 29
- ASX_SilenceDetector_GetEventEnable ASX_SYSTEM_TYPE_ALSA SubSystemTypes, 20
- SILENCEDETECTOR_SilenceDetector, 154
- ASX_SYSTEM_TYPE_ANY SubSystemTypes, 20
- ASX_SilenceDetector_GetThreshold ASX_SYSTEM_TYPE_ASIO SubSystemTypes, 20
- SILENCEDETECTOR_SilenceDetector, 155
- ASX_SYSTEM_TYPE_AVB_1722_1 SubSystemTypes, 20
- ASX_SilenceDetector_SetDelay ASX_SYSTEM_TYPE_COUNT SubSystemTypes, 20
- SILENCEDETECTOR_SilenceDetector, 155
- ASX_SYSTEM_TYPE_DIRECTX SubSystemTypes, 20
- ASX_SilenceDetector_SetEnable ASX_SYSTEM_TYPE_DUMMY SubSystemTypes, 21
- SILENCEDETECTOR_SilenceDetector, 155
- ASX_SYSTEM_TYPE_HPI SubSystemTypes, 21
- ASX_SilenceDetector_SetEventEnable ASX_SYSTEM_TYPE_HPIUDP SubSystemTypes, 21
- SILENCEDETECTOR_SilenceDetector, 156
- ASX_SYSTEM_TYPE_HPI SubSystemTypes, 21
- ASX_SilenceDetector_SetThreshold ASX_SYSTEM_TYPE_PCXTOOLS SubSystemTypes, 21
- SILENCEDETECTOR_SilenceDetector, 156
- ASX_SYSTEM_TYPE_PORTAUDIO SubSystemTypes, 21
- ASX_System_Create System, 23
- ASX_System_CreateSubSystem ASX_SYSTEM_TYPE_SNMP SubSystemTypes, 21
- System, 23
- ASX_System_Delete ASX_SYSTEM_TYPE_WAVE SubSystemTypes, 21
- System, 24
- ASX_System_GetAdapter ASX_TIME
- System, 24
- asx.h, 203
- ASX_System_GetAdapterCount ASX_ToneDetector_GetEnable
- System, 25
- TONEDETECTOR_ToneDetector, 151
- ASX_System_GetCobranetAutoassignParms ASX_ToneDetector_GetEventEnable
- System, 25
- TONEDETECTOR_ToneDetector, 151

-
- | | |
|--|-----------------------------------|
| ASX_ToneDetector_GetFrequency | ASX_Tuner_GetHdRadioDspVersion |
| TONEDETECTOR_ToneDetector, 151 | Tuner, 95 |
| ASX_ToneDetector_GetState | ASX_Tuner_GetHdRadioSdkVersion |
| TONEDETECTOR_ToneDetector, 151 | Tuner, 95 |
| ASX_ToneDetector_GetThreshold | ASX_Tuner_GetHdRadioSignalQuality |
| TONEDETECTOR_ToneDetector, 152 | Tuner, 96 |
| ASX_ToneDetector_SetEnable | ASX_Tuner_GetMode |
| TONEDETECTOR_ToneDetector, 152 | Tuner, 96 |
| ASX_ToneDetector_SetEventEnable | ASX_Tuner_GetProgram |
| TONEDETECTOR_ToneDetector, 152 | Tuner, 96 |
| ASX_ToneDetector_SetThreshold | ASX_Tuner_GetRawRFLevel |
| TONEDETECTOR_ToneDetector, 152 | Tuner, 97 |
| ASX_Tuner_EnumerateBand | ASX_Tuner_GetRFLevel |
| Tuner, 88 | Tuner, 97 |
| ASX_Tuner_EnumerateDeemphasis | ASX_Tuner_GetStatus |
| Tuner, 89 | Tuner, 97 |
| ASX_Tuner_EnumerateHdBlend | ASX_Tuner_SetBand |
| Tuner, 89 | Tuner, 98 |
| ASX_Tuner_EnumerateProgram | ASX_Tuner_SetDabAudioService |
| Tuner, 90 | Tuner, 98 |
| ASX_Tuner_GetBand | ASX_Tuner_SetDeemphasis |
| Tuner, 90 | Tuner, 98 |
| ASX_Tuner_GetDabAudioInfo | ASX_Tuner_SetFrequency |
| Tuner, 90 | Tuner, 99 |
| ASX_Tuner_GetDabAudioServiceCount | ASX_Tuner_SetGain |
| Tuner, 91 | Tuner, 99 |
| ASX_Tuner_GetDabAudioServiceName | ASX_Tuner_SetHdBlend |
| Tuner, 91 | Tuner, 100 |
| ASX_Tuner_GetDabMultiplexId | ASX_Tuner_SetMode |
| Tuner, 91 | Tuner, 100 |
| ASX_Tuner_GetDabMultiplexName | ASX_Tuner_SetProgram |
| Tuner, 92 | Tuner, 100 |
| ASX_Tuner_GetDabServiceId | ASX_Volume_GetChannels |
| Tuner, 92 | Volume, 78 |
| ASX_Tuner_GetDeemphasis | ASX_Volume_GetGain |
| Tuner, 92 | Volume, 78 |
| ASX_Tuner_GetDigitalSignalQuality | ASX_Volume_GetMute |
| Tuner, 93 | Volume, 79 |
| ASX_Tuner_GetFirmwareVersion | ASX_Volume_GetRange |
| Tuner, 93 | Volume, 79 |
| ASX_Tuner_GetFrequency | ASX_Volume_SetAutofade |
| Tuner, 93 | Volume, 80 |
| ASX_Tuner_GetFrequencyRange | ASX_Volume_SetGain |
| Tuner, 94 | Volume, 80 |
| ASX_Tuner_GetGain | ASX_Volume_SetMute |
| Tuner, 94 | Volume, 81 |
| ASX_Tuner_GetGainRange | ASX_Vox_GetLevel |
| Tuner, 94 | Vox, 117 |
| ASX_Tuner_GetHdBlend | ASX_Vox_GetRange |
| Tuner, 95 | Vox, 117 |
-

- ASX_Vox_SetLevel
 - Vox, [117](#)
- asxADAPTER_PROPERTY_ERRATA_1
 - asx.h, [203](#)
- asxADAPTER_PROPERTY_FIRMWARE_ID
 - asx.h, [203](#)
- asxADAPTER_PROPERTY_SSX2_SETTINGS
 - asx.h, [203](#)
- asxADAPTER_PROPERTY_SUPPORTS_SSX2
 - asx.h, [203](#)
- asxADAPTER_PROPERTY_SUPPORTS_FW_UPDATE
 - asx.h, [203](#)
- asxADAPTER_PROPERTY_SUPPORTS_SNMP
 - asx.h, [203](#)
- asxADAPTER_PROPERTY_SYNC_HEADER_CONNECTIONS
 - asx.h, [203](#)
- asxADAPTER_PROPERTY
 - asx.h, [203](#)
- asxADAPTERMODE
 - asx.h, [203](#)
- asxADAPTERMODE_12_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_16_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_1_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_24_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_32_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_4_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_6_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_8_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_9_PLAY
 - asx.h, [204](#)
- asxADAPTERMODE_ILLEGAL
 - asx.h, [204](#)
- asxADAPTERMODE_LOW_LATENCY
 - asx.h, [204](#)
- asxADAPTERMODE_MODE_1
 - asx.h, [204](#)
- asxADAPTERMODE_MODE_2
 - asx.h, [204](#)
- asxADAPTERMODE_MODE_3
 - asx.h, [204](#)
- asxADAPTERMODE_MONO
 - asx.h, [204](#)
- asxADAPTERMODE_MULTICHANNEL
 - asx.h, [204](#)
- asxADPROPENUM_MODE_PROPERTIES
 - asx.h, [204](#)
- asxADPROPENUM_MODE_SETTINGS
 - asx.h, [204](#)
- asxADPROPENUM_SSX2_OFF
 - asx.h, [204](#)
- asxADPROPENUM_SSX2_ON
 - asx.h, [204](#)
- asxADPROPENUM_MODE
 - asx.h, [204](#)
- asxADPROPENUM_SSX2
 - asx.h, [204](#)
- asxAESEBU_ERROR
 - asx.h, [205](#)
- asxAESEBU_ERROR_BIPHASE_VIOLATION
 - asx.h, [205](#)
- asxAESEBU_ERROR_CHANNELSTATUS_CRC
 - asx.h, [205](#)
- asxAESEBU_ERROR_NOT_LOCKED
 - asx.h, [205](#)
- asxAESEBU_ERROR_PARITY_ERROR
 - asx.h, [205](#)
- asxAESEBU_ERROR_POOR_QUALITY
 - asx.h, [205](#)
- asxAESEBU_ERROR_VALIDITY
 - asx.h, [205](#)
- asxAESEBU_FORMAT_AESEBU
 - asx.h, [205](#)
- asxAESEBU_FORMAT_SPDIF
 - asx.h, [205](#)
- asxAESEBU_FORMAT_UNDEFINED
 - asx.h, [205](#)
- asxAESEBU_FORMAT
 - asx.h, [204](#)
- asxAESEBU_STATUS
 - asx.h, [205](#)
- asxAUDIO_FORMAT_DOLBY_AC2
 - asx.h, [205](#)
- asxAUDIO_FORMAT_MPEG_AACPLUS
 - asx.h, [205](#)
- asxAUDIO_FORMAT_MPEG_L2
 - asx.h, [205](#)

- asxAUDIO_FORMAT_MPEG_L3
 - asx.h, [205](#)
- asxAUDIO_FORMAT_NONE
 - asx.h, [205](#)
- asxAUDIO_FORMAT_PCM16
 - asx.h, [205](#)
- asxAUDIO_FORMAT_PCM20
 - asx.h, [205](#)
- asxAUDIO_FORMAT_PCM24
 - asx.h, [205](#)
- asxAUDIO_FORMAT_PCM32
 - asx.h, [205](#)
- asxAUDIO_FORMAT_PCM32_FLOAT
 - asx.h, [205](#)
- asxAUDIO_FORMAT_PCM8
 - asx.h, [205](#)
- asxAUDIO_FORMAT
 - asx.h, [205](#)
- asxCHANNELMODE
 - asx.h, [205](#)
- asxCHANNELMODE_ILLEGAL
 - asx.h, [206](#)
- asxCHANNELMODE_LEFTTOSTEREO
 - asx.h, [206](#)
- asxCHANNELMODE_NORMAL
 - asx.h, [206](#)
- asxCHANNELMODE_RIGHTTOSTEREO
 - asx.h, [206](#)
- asxCHANNELMODE_STEREOTOLEFT
 - asx.h, [206](#)
- asxCHANNELMODE_STEREOTORIGHT
 - asx.h, [206](#)
- asxCHANNELMODE_SWAP
 - asx.h, [206](#)
- asxCOBANET_IFSTATUS_ACTIVE_CONNECTION
 - asx.h, [206](#)
- asxCOBANET_IFSTATUS_FULL_DUPLEX
 - asx.h, [206](#)
- asxCOBANET_IFSTATUS_LINK_ESTABLISHED
 - asx.h, [206](#)
- asxCOBANET_LATENCY_133ms
 - asx.h, [206](#)
- asxCOBANET_LATENCY_266ms
 - asx.h, [206](#)
- asxCOBANET_LATENCY_533ms
 - asx.h, [206](#)
- asxCOBANET_MODE_NETWORK
 - asx.h, [207](#)
- asxCOBANET_MODE_TETHERED
 - asx.h, [207](#)
- asxCOBANET_IFSTATUS
 - asx.h, [206](#)
- asxCOBANET_LATENCY
 - asx.h, [206](#)
- asxCOBANET_MODE
 - asx.h, [206](#)
- asxCobranetIpAutoassignParameters, [163](#)
 - addr_end, [163](#)
 - addr_start, [163](#)
 - autoassign, [163](#)
- asxCOMPANDER_INDEX_COMPANDER
 - asx.h, [207](#)
- asxCOMPANDER_INDEX_NOISEGATE
 - asx.h, [207](#)
- asxCOMPANDER_INDEX
 - asx.h, [207](#)
- asxCONTROL
 - asx.h, [207](#)
- asxCONTROL_AES18_BLOCK_GENERATOR
 - asx.h, [207](#)
- asxCONTROL_AES18_RECEIVER
 - asx.h, [207](#)
- asxCONTROL_AES18_TRANSMITTER
 - asx.h, [207](#)
- asxCONTROL_AESEBU_RECEIVER
 - asx.h, [207](#)
- asxCONTROL_AESEBU_TRANSMITTER
 - asx.h, [207](#)
- asxCONTROL_BIT_STREAM
 - asx.h, [208](#)
- asxCONTROL_BLOCK
 - asx.h, [208](#)
- asxCONTROL_CHANNEL_MODE
 - asx.h, [208](#)
- asxCONTROL_COBRANET
 - asx.h, [208](#)
- asxCONTROL_COBRANET_RECEIVER
 - asx.h, [208](#)
- asxCONTROL_COBRANET_TRANSMITTER
 - asx.h, [208](#)
- asxCONTROL_COMPANDER
 - asx.h, [208](#)
- asxCONTROL_CONNECTION
 - asx.h, [207](#)
- asxCONTROL_GENERIC
 - asx.h, [208](#)
- asxCONTROL_GPIO
 - asx.h, [208](#)
- asxCONTROL_INVALID
 - asx.h, [207](#)

- asxCNTRL_LAST_CNTRL
 - asx.h, [208](#)
- asxCNTRL_LEVEL
 - asx.h, [207](#)
- asxCNTRL_METER
 - asx.h, [207](#)
- asxCNTRL_MICROPHONE
 - asx.h, [208](#)
- asxCNTRL_MULTIPLEXER
 - asx.h, [207](#)
- asxCNTRL_MUTE
 - asx.h, [207](#)
- asxCNTRL_PAD
 - asx.h, [208](#)
- asxCNTRL_PARAMETRIC_EQ
 - asx.h, [208](#)
- asxCNTRL_PLAYER
 - asx.h, [208](#)
- asxCNTRL_RDS
 - asx.h, [207](#)
- asxCNTRL_RECORDER
 - asx.h, [208](#)
- asxCNTRL_RESERVED_525
 - asx.h, [208](#)
- asxCNTRL_RESERVED_526
 - asx.h, [208](#)
- asxCNTRL_RESERVED_527
 - asx.h, [208](#)
- asxCNTRL_RESERVED_528
 - asx.h, [208](#)
- asxCNTRL_SAMPLE_CLOCK
 - asx.h, [208](#)
- asxCNTRL_SILENCEDETECTOR
 - asx.h, [208](#)
- asxCNTRL_SRC
 - asx.h, [208](#)
- asxCNTRL_TONEDETECTOR
 - asx.h, [208](#)
- asxCNTRL_TUNER
 - asx.h, [207](#)
- asxCNTRL_VOLUME
 - asx.h, [207](#)
- asxCNTRL_VOX
 - asx.h, [207](#)
- asxEQBANDTYPE
 - asx.h, [208](#)
- asxEQBANDTYPE_BANDPASS
 - asx.h, [209](#)
- asxEQBANDTYPE_BANDSTOP
 - asx.h, [209](#)
- asxEQBANDTYPE_BYPASS
 - asx.h, [208](#)
- asxEQBANDTYPE_EQUALIZER
 - asx.h, [209](#)
- asxEQBANDTYPE_HIGHPASS
 - asx.h, [209](#)
- asxEQBANDTYPE_HIGHSHELF
 - asx.h, [208](#)
- asxEQBANDTYPE_LOWPASS
 - asx.h, [209](#)
- asxEQBANDTYPE_LOWSHELF
 - asx.h, [208](#)
- asxERROR
 - asx.h, [209](#)
- asxERROR_AES18
 - asx.h, [209](#)
- asxERROR_ALREADY_OPEN
 - asx.h, [209](#)
- asxERROR_ASXObject
 - asx.h, [209](#)
- asxERROR_BUFFER_TOO_SMALL
 - asx.h, [209](#)
- asxERROR_COBRANET_NODE_FOUND
 - asx.h, [210](#)
- asxERROR_COBRANET_NODE_NOT_FOUND
 - asx.h, [210](#)
- asxERROR_COBRANET_NODE_UNREACHABLE
 - asx.h, [210](#)
- asxERROR_COMMUNICATING_WITH_DEVICE
 - asx.h, [209](#)
- asxERROR_CONTROL_NOT_READY
 - asx.h, [210](#)
- asxERROR_DEPRECATED
 - asx.h, [210](#)
- asxERROR_DISCO_DLL_NOT_FOUND
 - asx.h, [210](#)
- asxERROR_DUPLICATE_ADAPTER_INDEX
 - asx.h, [210](#)
- asxERROR_ENUMERATE_INDEX_OUT_OF_RANGE
 - asx.h, [209](#)
- asxERROR_FILE_OPEN_FAILED
 - asx.h, [211](#)
- asxERROR_HOST_NOT_FOUND
 - asx.h, [210](#)
- asxERROR_INDEX_OUT_OF_RANGE
 - asx.h, [209](#)

asxERROR_INTERNAL_BUFFERING_- ERROR asx.h, 209	asxERROR_PLAYER_OUT_OF_SEQUENCE_- CALL asx.h, 211
asxERROR_INVALID_CONTROL asx.h, 210	asxERROR_PLAYER_TIME_OUT asx.h, 211
asxERROR_INVALID_CONTROL_ATTRIBUTE asx.h, 210	asxERROR_PLAYER_TWAV asx.h, 211
asxERROR_INVALID_CONTROL_NOT_- FOUND asx.h, 210	asxERROR_PLAYER_UNSUPPORTEDFORMAT asx.h, 211
asxERROR_INVALID_CONTROL_OPERATION asx.h, 210	asxERROR_RECORDER_FILECREATEERROR asx.h, 211
asxERROR_INVALID_CONTROL_VALUE asx.h, 210	asxERROR_RECORDER_FILEWRITEERROR asx.h, 211
asxERROR_INVALID_FORMAT asx.h, 209	asxERROR_RECORDER_FORMATMISMATCH asx.h, 211
asxERROR_INVALID_NUMBER_OF_CHANNELS asx.h, 210	asxERROR_RECORDER_INTERNAL_- STATE_FAILURE asx.h, 211
asxERROR_INVALID_OPERATION asx.h, 209	asxERROR_RECORDER_INVALIDFILENAME asx.h, 211
asxERROR_IP_ASSIGNED asx.h, 210	asxERROR_RECORDER_OUT_OF_SEQUENCE_- CALL asx.h, 211
asxERROR_IP_AUTOASSIGN_DISABLED asx.h, 210	asxERROR_RECORDER_TIME_OUT asx.h, 211
asxERROR_IP_CHANGED asx.h, 210	asxERROR_RECORDER_TWAV asx.h, 211
asxERROR_MIXER_SAVECONTROLSTATE asx.h, 211	asxERROR_STARTING_DEVICE asx.h, 209
asxERROR_NO_ERROR asx.h, 209	asxERROR_TOO_MANY_CLIENTS asx.h, 210
asxERROR_NO_IP_ADDRESSES_AVAILABLE asx.h, 210	asxERROR_UNIMPLEMENTED asx.h, 209
asxERROR_NOT_OPEN asx.h, 209	asxERROR_UNKNOWN asx.h, 211
asxERROR_OUTOFMEMORY asx.h, 210	asxERROR_UNSUPPORTED_CONTROL_- ATTRIBUTE asx.h, 210
asxERROR_PCAP_ERROR asx.h, 210	asxFILE_FORMAT_RAW asx.h, 212
asxERROR_PLAYER_FILEOPENERERROR asx.h, 211	asxFILE_FORMAT_WAV asx.h, 212
asxERROR_PLAYER_FILEREADERERROR asx.h, 211	asxFILE_MODE_APPEND asx.h, 212
asxERROR_PLAYER_INTERNAL_STATE_- FAILURE asx.h, 211	asxFILE_MODE_CREATE asx.h, 212
asxERROR_PLAYER_INVALIDFILEFORMAT asx.h, 211	asxFILE_FORMAT asx.h, 211
asxERROR_PLAYER_NOFILE asx.h, 211	asxFILE_MODE asx.h, 212

- asxHANDLE_ADAPTER
 - asx.h, [212](#)
- asxHANDLE_CONTROL
 - asx.h, [212](#)
- asxHANDLE_INVALID
 - asx.h, [212](#)
- asxHANDLE_LAST
 - asx.h, [212](#)
- asxHANDLE_MIXER
 - asx.h, [212](#)
- asxHANDLE_NODE
 - asx.h, [212](#)
- asxHANDLE_SYSTEM
 - asx.h, [212](#)
- asxHANDLE_TYPE
 - asx.h, [212](#)
- asxMETER_PEAK
 - asx.h, [212](#)
- asxMETER_RMS
 - asx.h, [212](#)
- asxMETER_TYPE
 - asx.h, [212](#)
- asxMSG_LOGGING_DEBUG
 - asx.h, [213](#)
- asxMSG_LOGGING_DISABLE
 - asx.h, [213](#)
- asxMSG_LOGGING_ERROR
 - asx.h, [213](#)
- asxMSG_LOGGING_INFO
 - asx.h, [213](#)
- asxMSG_LOGGING_NOTICE
 - asx.h, [213](#)
- asxMSG_LOGGING_VERBOSE
 - asx.h, [213](#)
- asxMSG_LOGGING_WARNING
 - asx.h, [213](#)
- asxMSG_LOGGING
 - asx.h, [212](#)
- asxNODE
 - asx.h, [213](#)
- asxNODE_ADAPTER
 - asx.h, [213](#)
- asxNODE_AESEBU_IN
 - asx.h, [213](#)
- asxNODE_AESEBU_OUT
 - asx.h, [214](#)
- asxNODE_ANALOG_IN
 - asx.h, [214](#)
- asxNODE_ANALOG_OUT
 - asx.h, [214](#)
- asxNODE_AVB_IN_AUDIO
 - asx.h, [214](#)
- asxNODE_AVB_IN_STREAM
 - asx.h, [214](#)
- asxNODE_AVB_OUT_AUDIO
 - asx.h, [214](#)
- asxNODE_AVB_OUT_STREAM
 - asx.h, [214](#)
- asxNODE_BITSTREAM_IN
 - asx.h, [213](#)
- asxNODE_BLULINK_IN
 - asx.h, [214](#)
- asxNODE_BLULINK_OUT
 - asx.h, [214](#)
- asxNODE_CLOCK_SOURCE_IN
 - asx.h, [213](#)
- asxNODE_COBRANET_IN
 - asx.h, [213](#)
- asxNODE_COBRANET_OUT
 - asx.h, [214](#)
- asxNODE_COBRANET_RECEIVER
 - asx.h, [213](#)
- asxNODE_COBRANET_TRANSMITTER
 - asx.h, [214](#)
- asxNODE_FIRST_DEST_NODE
 - asx.h, [214](#)
- asxNODE_INTERNAL_IN
 - asx.h, [214](#)
- asxNODE_INTERNAL_OUT
 - asx.h, [214](#)
- asxNODE_INVALID
 - asx.h, [213](#)
- asxNODE_LAST_DEST_NODE
 - asx.h, [214](#)
- asxNODE_LAST_SOURCE_NODE
 - asx.h, [214](#)
- asxNODE_LINE_IN
 - asx.h, [213](#)
- asxNODE_LINE_OUT
 - asx.h, [214](#)
- asxNODE_MICROPHONE_IN
 - asx.h, [213](#)
- asxNODE_NONE
 - asx.h, [213](#)
- asxNODE_PLAYER
 - asx.h, [213](#)
- asxNODE_RADIO_FREQ_IN
 - asx.h, [213](#)
- asxNODE_RADIO_FREQ_OUT
 - asx.h, [214](#)

- asxNODE_RECORDER
 - asx.h, [214](#)
- asxNODE_RTP_DESTINATION_IN
 - asx.h, [214](#)
- asxNODE_RTP_SOURCE_OUT
 - asx.h, [214](#)
- asxNODE_SDI_IN
 - asx.h, [214](#)
- asxNODE_SDI_OUT
 - asx.h, [214](#)
- asxNODE_SPEAKER_OUT
 - asx.h, [214](#)
- asxNODE_TUNER_IN
 - asx.h, [213](#)
- asxPARAM_FLAG_READABLE
 - asx.h, [221](#)
- asxPARAM_FLAG_VOLATILE
 - asx.h, [221](#)
- asxPARAM_FLAG_WRITEABLE
 - asx.h, [221](#)
- asxPARAM_RANGE_ENUMERATED
 - asx.h, [222](#)
- asxPARAM_RANGE_ENUMERATED_-
 - FLOAT
 - asx.h, [222](#)
 - INTEGER
 - asx.h, [222](#)
- asxPARAM_RANGE_NONE
 - asx.h, [222](#)
- asxPARAM_RANGE_NUMBER_OF_BITS
 - asx.h, [222](#)
- asxPARAM_RANGE_STEPPED_FLOAT
 - asx.h, [222](#)
- asxPARAM_RANGE_STEPPED_INTEGER
 - asx.h, [222](#)
- asxPARAM_RANGE_STRING_LENGTH
 - asx.h, [222](#)
- asxPARAM_TYPE_BOOLEAN
 - asx.h, [221](#)
- asxPARAM_TYPE_DOUBLE
 - asx.h, [221](#)
- asxPARAM_TYPE_FLOAT
 - asx.h, [221](#)
- asxPARAM_TYPE_INTEGER
 - asx.h, [221](#)
- asxPARAM_TYPE_IP4_ADDRESS
 - asx.h, [221](#)
- asxPARAM_TYPE_IP6_ADDRESS
 - asx.h, [221](#)
- asxPARAM_TYPE_MAC_ADDRESS
 - asx.h, [221](#)
- asxPARAM_TYPE_NONE
 - asx.h, [221](#)
- asxPARAM_TYPE_STRING
 - asx.h, [221](#)
- asxParameterRangeInfo, [163](#)
 - count, [165](#)
 - enumerated, [165](#)
 - enumerated_float, [165](#)
 - enumerated_integer, [165](#)
 - enums, [165](#)
 - floating, [165](#)
 - fmax, [165](#)
 - fmin, [165](#)
 - fstep, [165](#)
 - integer, [165](#)
 - max, [165](#)
 - max_len, [165](#)
 - min, [165](#)
 - step, [165](#)
 - string, [165](#)
 - type, [165](#)
 - u, [165](#)
 - value, [165](#)
- asxParameterRangeInfo_NamedEnumerated,
 - [166](#)
 - name, [166](#)
 - value, [166](#)
- asxParameterValue, [166](#)
 - eType, [166](#)
 - size, [166](#)
 - uItems, [166](#)
 - value, [166](#)
- asxPLAYER_DESTROY
 - asx.h, [215](#)
- asxPLAYER_DONE
 - asx.h, [215](#)
- asxPLAYER_FILE_COMPLETE
 - asx.h, [214](#)
- asxPLAYER_FILE_START
 - asx.h, [214](#)
- asxPLAYER_FILELIST_COMPLETE
 - asx.h, [214](#)
- asxPLAYER_INIT
 - asx.h, [215](#)
- asxPLAYER_OPEN
 - asx.h, [215](#)
- asxPLAYER_PAUSED
 - asx.h, [215](#)

- asxPLAYER_PREFILL
 - asx.h, [215](#)
- asxPLAYER_RUNNING
 - asx.h, [215](#)
- asxPLAYER_FLAGS
 - asx.h, [214](#)
- asxPLAYER_STATE
 - asx.h, [214](#)
- asxRECORD_MODE_DONT_CARE
 - asx.h, [215](#)
- asxRECORD_MODE_DUAL_MONO
 - asx.h, [215](#)
- asxRECORD_MODE_JOINT_STEREO
 - asx.h, [215](#)
- asxRECORD_MODE_MONO
 - asx.h, [215](#)
- asxRECORD_MODE_STEREO
 - asx.h, [215](#)
- asxRECORD_MODE
 - asx.h, [215](#)
- asxRECORDER_DESTROY
 - asx.h, [215](#)
- asxRECORDER_DONE
 - asx.h, [215](#)
- asxRECORDER_INIT
 - asx.h, [215](#)
- asxRECORDER_OPEN
 - asx.h, [215](#)
- asxRECORDER_PAUSED
 - asx.h, [215](#)
- asxRECORDER_RUNNING
 - asx.h, [215](#)
- asxRECORDER_STATE
 - asx.h, [215](#)
- asxSAMPLE_CLOCK_SOURCE_ADAPTER
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT0
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT1
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT2
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT3
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT4
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT5
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT6
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT7
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT8
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT15
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT16
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT17
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT18
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT19
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT2
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT20
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT21
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT22
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT23
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT24
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT25
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT26
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT27
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT28
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT29
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT3
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT30
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT31
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT32
 - asx.h, [217](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT4
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT5
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT6
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT7
 - asx.h, [216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBUINPUT8
 - asx.h, [216](#)

- asxSAMPLE_CLOCK_SOURCE_AESEBU - [asxstring.h, 216](#)
- asxSAMPLE_CLOCK_SOURCE_AESEBU - [asxstring.h, 216](#)
- asxSAMPLE_CLOCK_SOURCE_BLULINK - [asxstring.h, 218](#)
- asxSAMPLE_CLOCK_SOURCE_LIVEWIRE - [asxstring.h, 218](#)
- asxSAMPLE_CLOCK_SOURCE_LOCAL - [asxstring.h, 217](#)
- asxSAMPLE_CLOCK_SOURCE_NETWORK - [asxstring.h, 216](#)
- asxSAMPLE_CLOCK_SOURCE_PREV - [asxstring.h, 217](#)
- asxSAMPLE_CLOCK_SOURCE_SMPTE - [asxstring.h, 216](#)
- asxSAMPLE_CLOCK_SOURCE_UNDEFINED - [asxstring.h, 217](#)
- asxSAMPLE_CLOCK_SOURCE_WORD - [asxstring.h, 216](#)
- asxSAMPLE_CLOCK_SOURCE_WORD - [asxstring.h, 216](#)
- asxSAMPLE_RATE_11025 - [asx.h, 218](#)
- asxSAMPLE_RATE_12000 - [asx.h, 218](#)
- asxSAMPLE_RATE_16000 - [asx.h, 218](#)
- asxSAMPLE_RATE_176400 - [asx.h, 218](#)
- asxSAMPLE_RATE_192000 - [asx.h, 218](#)
- asxSAMPLE_RATE_22050 - [asx.h, 218](#)
- asxSAMPLE_RATE_24000 - [asx.h, 218](#)
- asxSAMPLE_RATE_32000 - [asx.h, 218](#)
- asxSAMPLE_RATE_44100 - [asx.h, 218](#)
- asxSAMPLE_RATE_48000 - [asx.h, 218](#)
- asxSAMPLE_RATE_64000 - [asx.h, 218](#)
- asxSAMPLE_RATE_8000 - [asx.h, 218](#)
- asxSAMPLE_RATE_88200 - [asx.h, 218](#)
- asxSAMPLE_RATE_96000 - [asx.h, 218](#)
- asxSAMPLE_RATE_UNDEFINED - [asx.h, 218](#)
- asxSAMPLE_CLOCK_SOURCE - [asx.h, 215](#)
- asxSAMPLE_RATE - [asx.h, 218](#)
- asxstring.h, 222 - [asxstring.h, 222](#)
- ASX32_API, 223 - [asxstring.h, 223](#)
- ASXSTRING_EnumToString, 223 - [asxstring.h, 223](#)
- ASXSTRING_StringToEnum, 223 - [asxstring.h, 223](#)
- ASXSTRING_EnumToString - [asxstring.h, 223](#)
- ASXSTRING_StringToEnum - [asxstring.h, 223](#)
- asxTIMESCALE - [asx.h, 218](#)
- asxTIMESCALE_BYTES - [asx.h, 218](#)
- asxTIMESCALE_BYTES_REMAINING - [asx.h, 218](#)
- asxTIMESCALE_INVALID - [asx.h, 218](#)
- asxTIMESCALE_MILLISECONDS - [asx.h, 218](#)
- asxTIMESCALE_MILLISECONDS_REMAINING - [asx.h, 218](#)
- asxTIMESCALE_SAMPLES - [asx.h, 218](#)
- asxTIMESCALE_SAMPLES_REMAINING - [asx.h, 219](#)
- asxTUNER_RDS_TYPE_RBDS - [asx.h, 219](#)
- asxTUNER_RDS_TYPE_RDS - [asx.h, 219](#)
- asxTUNER_STATUS_DIGITAL - [asx.h, 219](#)
- asxTUNER_STATUS_FIRMWARE_LOADING - [asx.h, 219](#)
- asxTUNER_STATUS_FM_STEREO - [asx.h, 219](#)
- asxTUNER_STATUS_MULTIPROGRAM - [asx.h, 219](#)
- asxTUNER_STATUS_PLL_LOCKED - [asx.h, 219](#)
- asxTUNER_STATUS_VIDEO_COLOR - [asx.h, 219](#)
- asxTUNER_STATUS_VIDEO_COLOR - [asx.h, 219](#)

- asxTUNER_STATUS_VIDEO_HORZ_SYNC_- asx.h, [220](#)
- MISSING
 - asx.h, [219](#)
- asxTUNER_STATUS_VIDEO_IS_60HZ
 - asx.h, [219](#)
- asxTUNER_STATUS_VIDEO_VALID
 - asx.h, [219](#)
- asxTUNER_RDS_TYPE
 - asx.h, [219](#)
- asxTUNER_STATUS
 - asx.h, [219](#)
- asxTUNERBAND
 - asx.h, [219](#)
- asxTUNERBAND_AM
 - asx.h, [219](#)
- asxTUNERBAND_AUX
 - asx.h, [219](#)
- asxTUNERBAND_DAB
 - asx.h, [220](#)
- asxTUNERBAND_FM
 - asx.h, [219](#)
- asxTUNERBAND_FM_STEREO
 - asx.h, [219](#)
- asxTUNERBAND_TV
 - asx.h, [219](#)
- asxTUNERBAND_TV_PAL_BG
 - asx.h, [220](#)
- asxTUNERBAND_TV_PAL_DK
 - asx.h, [220](#)
- asxTUNERBAND_TV_PAL_I
 - asx.h, [220](#)
- asxTUNERBAND_TV_SECAM_L
 - asx.h, [220](#)
- asxTUNERDEEMPHASIS
 - asx.h, [220](#)
- asxTUNERDEEMPHASIS_50
 - asx.h, [220](#)
- asxTUNERDEEMPHASIS_75
 - asx.h, [220](#)
- asxTUNERDEEMPHASIS_NONE
 - asx.h, [220](#)
- asxTUNERHDBLEND
 - asx.h, [220](#)
- asxTUNERHDBLEND_ANALOG
 - asx.h, [220](#)
- asxTUNERHDBLEND_AUTO
 - asx.h, [220](#)
- asxTUNERMODE
 - asx.h, [220](#)
- asxTUNERMODE_RSS
 - asxTUNERMODE_RSS_DISABLE
 - asx.h, [220](#)
 - asxTUNERMODE_RSS_ENABLE
 - asx.h, [220](#)
 - asxTUNERPROGRAM
 - asx.h, [220](#)
 - asxTUNERPROGRAM_1
 - asx.h, [220](#)
 - asxTUNERPROGRAM_2
 - asx.h, [221](#)
 - asxTUNERPROGRAM_3
 - asx.h, [221](#)
 - asxTUNERPROGRAM_4
 - asx.h, [221](#)
 - asxTUNERPROGRAM_5
 - asx.h, [221](#)
 - asxTUNERPROGRAM_6
 - asx.h, [221](#)
 - asxTUNERPROGRAM_7
 - asx.h, [221](#)
 - asxTUNERPROGRAM_8
 - asx.h, [221](#)
 - asxTUNERPROGRAM_NONE
 - asx.h, [220](#)
 - asxUCONTROL_PFLAGS
 - asx.h, [221](#)
 - asxUCONTROL_PTYPE
 - asx.h, [221](#)
 - asxUCONTROL_RTYPE
 - asx.h, [221](#)
 - asxVOLUME_AUTOFADE_LINEAR
 - asx.h, [222](#)
 - asxVOLUME_AUTOFADE_LOG
 - asx.h, [222](#)
 - asxVOLUME_AUTOFADE
 - asx.h, [222](#)
- autoassign
 - asxCobranetIpAutoassignParameters, [163](#)
- Block functions., [156](#)
- BLOCK_Block
 - ASX_Block_GetInfo, [157](#)
 - ASX_Block_Parameter_Get, [157](#)
 - ASX_Block_Parameter_GetElementCount, [158](#)
 - ASX_Block_Parameter_GetEnumName, [158](#)
 - ASX_Block_Parameter_GetFlags, [159](#)

- ASX_Block_Parameter_GetName, [159](#)
- ASX_Block_Parameter_GetRange, [159](#)
- ASX_Block_Parameter_GetType, [160](#)
- ASX_Block_Parameter_GetUnits, [160](#)
- ASX_Block_Parameter_Set, [161](#)
- Channel Mode control functions, [84](#)
- CHANNEL_MODE_Mode
 - ASX_ChannelMode_Enumerate, [85](#)
 - ASX_ChannelMode_Get, [85](#)
 - ASX_ChannelMode_Set, [85](#)
- Cobranet
 - ASX_Cobranet_EnumerateModes, [130](#)
 - ASX_Cobranet_GetConductorPriority, [130](#)
 - ASX_Cobranet_GetConductorStatus, [130](#)
 - ASX_Cobranet_GetDescription, [131](#)
 - ASX_Cobranet_GetErrorInfo, [131](#)
 - ASX_Cobranet_GetFirmwareRevision, [131](#)
 - ASX_Cobranet_GetIfStatus, [132](#)
 - ASX_Cobranet_GetIPAddress, [132](#)
 - ASX_Cobranet_GetLatencyAndSampleRate, [133](#)
 - ASX_Cobranet_GetLocation, [133](#)
 - ASX_Cobranet_GetMACAddress, [133](#)
 - ASX_Cobranet_GetMode, [134](#)
 - ASX_Cobranet_GetName, [134](#)
 - ASX_Cobranet_GetPersistence, [134](#)
 - ASX_Cobranet_GetSerialConfig, [135](#)
 - ASX_Cobranet_GetSerialEnable, [135](#)
 - ASX_Cobranet_GetStaticIPAddress, [135](#)
 - ASX_Cobranet_SetConductorPriority, [136](#)
 - ASX_Cobranet_SetIPAddress, [136](#)
 - ASX_Cobranet_SetLatencyAndSampleRate, [136](#)
 - ASX_Cobranet_SetLocation, [137](#)
 - ASX_Cobranet_SetMode, [137](#)
 - ASX_Cobranet_SetName, [137](#)
 - ASX_Cobranet_SetPersistence, [138](#)
 - ASX_Cobranet_SetSerialConfig, [138](#)
 - ASX_Cobranet_SetSerialEnable, [138](#)
 - ASX_Cobranet_SetStaticIPAddress, [138](#)
- CobraNet control functions, [128](#)
- Cobranet receiver control functions, [144](#)
- Cobranet transmitter control functions, [139](#)
- COBRANET_RECEIVER_CobranetRx
 - ASX_CobranetRx_GetBundle, [145](#)
 - ASX_CobranetRx_GetChannelMap, [146](#)
 - ASX_CobranetRx_GetMinimumDelay, [146](#)
 - ASX_CobranetRx_GetSourceMAC, [146](#)
 - ASX_CobranetRx_GetStatus, [147](#)
 - ASX_CobranetRx_SetBundle, [148](#)
 - ASX_CobranetRx_SetChannelMap, [148](#)
 - ASX_CobranetRx_SetMinimumDelay, [149](#)
 - ASX_CobranetRx_SetSourceMAC, [149](#)
- COBRANET_TRANSMITTER_CobranetTx
 - ASX_CobranetTx_GetBundle, [140](#)
 - ASX_CobranetTx_GetChannelCount, [140](#)
 - ASX_CobranetTx_GetChannelMap, [141](#)
 - ASX_CobranetTx_GetFormat, [141](#)
 - ASX_CobranetTx_GetStatus, [142](#)
 - ASX_CobranetTx_GetUnicastMode, [142](#)
 - ASX_CobranetTx_SetBundle, [142](#)
 - ASX_CobranetTx_SetChannelCount, [143](#)
 - ASX_CobranetTx_SetChannelMap, [143](#)
 - ASX_CobranetTx_SetFormat, [143](#)
 - ASX_CobranetTx_SetUnicastMode, [144](#)
- Compander
 - ASX_Compander_Get, [123](#)
 - ASX_Compander_GetAttackTimeConstant, [123](#)
 - ASX_Compander_GetDecayTimeConstant, [124](#)
 - ASX_Compander_GetEnable, [124](#)
 - ASX_Compander_GetMakeupGain, [124](#)
 - ASX_Compander_GetRatio, [125](#)
 - ASX_Compander_GetThreshold, [125](#)
 - ASX_Compander_Set, [125](#)
 - ASX_Compander_SetAttackTimeConstant, [126](#)
 - ASX_Compander_SetDecayTimeConstant, [126](#)
 - ASX_Compander_SetEnable, [127](#)
 - ASX_Compander_SetMakeupGain, [127](#)
 - ASX_Compander_SetRatio, [127](#)
 - ASX_Compander_SetThreshold, [128](#)
- Compander control functions, [122](#)
- Control generic functions, [54](#)
- CONTROL_ControlBase
 - ASX_Control_GetDestinationNode, [55](#)
 - ASX_Control_GetHpiControl, [55](#)
 - ASX_Control_GetSourceNode, [55](#)

- ASX_Control_GetSubSystem, [56](#)
- ASX_Control_GetType, [56](#)
- count
 - asxParameterRangeInfo, [165](#)
- enumerated
 - asxParameterRangeInfo, [165](#)
- enumerated_float
 - asxParameterRangeInfo, [165](#)
- enumerated_integer
 - asxParameterRangeInfo, [165](#)
- enums
 - asxParameterRangeInfo, [165](#)
- Error functions, [30](#)
- ERROR_Base
 - ASX_Error_Clear, [31](#)
 - ASX_Error_GetLast, [31](#)
 - ASX_Error_GetLastString, [32](#)
- eType
 - asxParameterValue, [166](#)
- floating
 - asxParameterRangeInfo, [165](#)
- fmax
 - asxParameterRangeInfo, [165](#)
- fmin
 - asxParameterRangeInfo, [165](#)
- fstep
 - asxParameterRangeInfo, [165](#)
- Generic control functions, [118](#)
- GENERIC_GenericControl
 - ASX_GetGenericControlName, [118](#)
- Gpio
 - ASX_GPIO_GetProperties, [114](#)
 - ASX_GPIO_InputGet, [115](#)
 - ASX_GPIO_OutputGet, [115](#)
 - ASX_GPIO_OutputSet, [116](#)
- GPIO control functions, [114](#)
- Handle
 - ASX_Handle_GetType, [30](#)
- Handle functions, [30](#)
- integer
 - asxParameterRangeInfo, [165](#)
- Level
 - ASX_Level_Get, [81](#)
 - ASX_Level_GetRange, [82](#)
 - ASX_Level_Set, [82](#)
- Level control functions, [81](#)
- max
 - asxParameterRangeInfo, [165](#)
- max_len
 - asxParameterRangeInfo, [165](#)
- Meter
 - ASX_Meter_GetBallistics, [74](#)
 - ASX_Meter_GetChannels, [75](#)
 - ASX_Meter_GetPeak, [75](#)
 - ASX_Meter_GetRMS, [76](#)
 - ASX_Meter_SetBallistics, [76](#)
- Meter control functions, [74](#)
- Microphone control functions, [118](#)
- MICROPHONE_Mic
 - ASX_Mic_GetPhantomPower, [119](#)
 - ASX_Mic_SetPhantomPower, [119](#)
- min
 - asxParameterRangeInfo, [165](#)
- Mixer
 - ASX_Mixer_GetBlockControlByNodeTypeAndIndex, [45](#)
 - ASX_Mixer_GetControl, [45](#)
 - ASX_Mixer_GetControlByNode, [46](#)
 - ASX_Mixer_GetControlByNodeTypeAndIndex, [46](#)
 - ASX_Mixer_GetControlCount, [47](#)
 - ASX_Mixer_GetDestinationNode, [48](#)
 - ASX_Mixer_GetDestinationNodeCount, [48](#)
 - ASX_Mixer_GetNodeByType, [48](#)
 - ASX_Mixer_GetNodeTypeCount, [49](#)
 - ASX_Mixer_GetSourceNode, [49](#)
 - ASX_Mixer_GetSourceNodeCount, [50](#)
 - ASX_Mixer_ResetControls, [50](#)
- Mixer functions, [44](#)
- Multiplexer control functions, [82](#)
- MULTIPLEXER_Mux
 - ASX_Multiplexer_Enumerate, [83](#)
 - ASX_Multiplexer_Get, [83](#)
 - ASX_Multiplexer_Set, [84](#)
- name
 - asxParameterRangeInfo_NamedEnumerated, [166](#)
- Node
 - ASX_Mixer_GetNodeIndex, [52](#)
 - ASX_Mixer_GetNodeType, [52](#)
 - ASX_Node_GetIndex, [52](#)
 - ASX_Node_GetLocation, [52](#)

- ASX_Node_GetName, [53](#)
- ASX_Node_GetSubSystem, [53](#)
- ASX_Node_GetType, [53](#)
- Node functions, [51](#)
- Pad
 - ASX_PAD_GetArtist, [101](#)
 - ASX_PAD_GetChannelName, [102](#)
 - ASX_PAD_GetComment, [102](#)
 - ASX_PAD_GetProgramType, [103](#)
 - ASX_PAD_GetProgramTypeString, [103](#)
 - ASX_PAD_GetRdsPI, [104](#)
 - ASX_PAD_GetTitle, [104](#)
- PAD control functions, [101](#)
- Parametric Equalizer control functions, [119](#)
- PARAMETRIC_EQ_ParametricEQ
 - ASX_EQ_GetBand, [120](#)
 - ASX_EQ_GetInfo, [120](#)
 - ASX_EQ_SetBand, [121](#)
 - ASX_EQ_SetState, [121](#)
- pcxport.txt, [223](#)
- Player
 - ASX_Player_Close, [59](#)
 - ASX_Player_Format_GetDetails, [59](#)
 - ASX_Player_Format_GetString, [60](#)
 - ASX_Player_GetFilename, [60](#)
 - ASX_Player_GetLoopMode, [61](#)
 - ASX_Player_GetPosition, [61](#)
 - ASX_Player_GetState, [61](#)
 - ASX_Player_GetTimeScale, [62](#)
 - ASX_Player_Open, [62](#)
 - ASX_Player_OpenPlaylist, [63](#)
 - ASX_Player_Pause, [64](#)
 - ASX_Player_PlaylistStatus, [64](#)
 - ASX_Player_PlaylistWait, [65](#)
 - ASX_Player_PreLoad, [65](#)
 - ASX_Player_RegisterCallback, [66](#)
 - ASX_Player_SetLoopMode, [66](#)
 - ASX_Player_SetPosition, [66](#)
 - ASX_Player_SetTimeScale, [67](#)
 - ASX_Player_Start, [67](#)
 - ASX_Player_Stop, [67](#)
 - ASX_Player_Wait, [68](#)
- Player control functions, [57](#)
- Recorder
 - ASX_Recorder_Close, [70](#)
 - ASX_Recorder_EnumerateFormat, [70](#)
 - ASX_Recorder_GetFilename, [71](#)
 - ASX_Recorder_GetPosition, [71](#)
 - ASX_Recorder_GetState, [72](#)
 - ASX_Recorder_Open, [72](#)
 - ASX_Recorder_Pause, [73](#)
 - ASX_Recorder_Start, [73](#)
 - ASX_Recorder_Stop, [73](#)
 - Recorder control functions, [68](#)
 - Sample clock control functions, [105](#)
 - SAMPLE_CLOCK_SampleClock
 - ASX_SampleClock_EnumerateClockSource, [106](#)
 - ASX_SampleClock_EnumerateLocalRate, [106](#)
 - ASX_SampleClock_EnumerateSampleRate, [106](#)
 - ASX_SampleClock_GetAutoSource, [106](#)
 - ASX_SampleClock_GetClockSource, [107](#)
 - ASX_SampleClock_GetLocalRate, [107](#)
 - ASX_SampleClock_GetLocalRateLock, [107](#)
 - ASX_SampleClock_GetSampleRate, [108](#)
 - ASX_SampleClock_SetAutoSource, [108](#)
 - ASX_SampleClock_SetClockSource, [108](#)
 - ASX_SampleClock_SetLocalRate, [109](#)
 - ASX_SampleClock_SetLocalRateLock, [109](#)
 - ASX_SampleClock_SetSampleRate, [110](#)
 - Silence detector control functions, [153](#)
 - SILENCEDETECTOR_SilenceDetector
 - ASX_SilenceDetector_GetDelay, [154](#)
 - ASX_SilenceDetector_GetEnable, [154](#)
 - ASX_SilenceDetector_GetEventEnable, [154](#)
 - ASX_SilenceDetector_GetState, [154](#)
 - ASX_SilenceDetector_GetThreshold, [155](#)
 - ASX_SilenceDetector_SetDelay, [155](#)
 - ASX_SilenceDetector_SetEnable, [155](#)
 - ASX_SilenceDetector_SetEventEnable, [156](#)
 - ASX_SilenceDetector_SetThreshold, [156](#)
 - size
 - asxParameterValue, [166](#)
 - step
 - asxParameterRangeInfo, [165](#)

- string
 - asxParameterRangeInfo, 165
- SubSystem types, 19
- SubSystemTypes
 - ASX_SYSTEM_TYPE_ALSA, 20
 - ASX_SYSTEM_TYPE_ANY, 20
 - ASX_SYSTEM_TYPE_ASIO, 20
 - ASX_SYSTEM_TYPE_AVB_1722_1, 20
 - ASX_SYSTEM_TYPE_COUNT, 20
 - ASX_SYSTEM_TYPE_DIRECTX, 20
 - ASX_SYSTEM_TYPE_DUMMY, 21
 - ASX_SYSTEM_TYPE_HPI, 21
 - ASX_SYSTEM_TYPE_HPIUDP, 21
 - ASX_SYSTEM_TYPE_PCXTOOLS, 21
 - ASX_SYSTEM_TYPE_PORTAUDIO, 21
 - ASX_SYSTEM_TYPE_SNMP, 21
 - ASX_SYSTEM_TYPE_WAVE, 21
- System
 - ASX_System_Create, 23
 - ASX_System_CreateSubSystem, 23
 - ASX_System_Delete, 24
 - ASX_System_GetAdapter, 24
 - ASX_System_GetAdapterCount, 25
 - ASX_System_GetCobranetAutoassignParams, 25
 - ASX_System_GetMessageLogging, 26
 - ASX_System_GetName, 26
 - ASX_System_GetVersion, 27
 - ASX_System_RegisterErrorCallback, 27
 - ASX_System_SetAvdeccInterface, 28
 - ASX_System_SetCobranetAutoassignParams, 28
 - ASX_System_SetHostNetworkInterface, 29
 - ASX_System_SetMessageLogging, 29
 - ASX_System_SupportsSubSystem, 29
- System functions, 22
- Tone detector control functions, 150
- TONEDETECTOR_ToneDetector
 - ASX_ToneDetector_GetEnable, 151
 - ASX_ToneDetector_GetEventEnable, 151
 - ASX_ToneDetector_GetFrequency, 151
 - ASX_ToneDetector_GetState, 151
 - ASX_ToneDetector_GetThreshold, 152
 - ASX_ToneDetector_SetEnable, 152
 - ASX_ToneDetector_SetEventEnable, 152
 - ASX_ToneDetector_SetThreshold, 152
- Tuner
 - ASX_Tuner_EnumerateBand, 88
 - ASX_Tuner_EnumerateDeemphasis, 89
 - ASX_Tuner_EnumerateHdBlend, 89
 - ASX_Tuner_EnumerateProgram, 90
 - ASX_Tuner_GetBand, 90
 - ASX_Tuner_GetDabAudioInfo, 90
 - ASX_Tuner_GetDabAudioServiceCount, 91
 - ASX_Tuner_GetDabAudioServiceName, 91
 - ASX_Tuner_GetDabMultiplexId, 91
 - ASX_Tuner_GetDabMultiplexName, 92
 - ASX_Tuner_GetDabServiceId, 92
 - ASX_Tuner_GetDeemphasis, 92
 - ASX_Tuner_GetDigitalSignalQuality, 93
 - ASX_Tuner_GetFirmwareVersion, 93
 - ASX_Tuner_GetFrequency, 93
 - ASX_Tuner_GetFrequencyRange, 94
 - ASX_Tuner_GetGain, 94
 - ASX_Tuner_GetGainRange, 94
 - ASX_Tuner_GetHdBlend, 95
 - ASX_Tuner_GetHdRadioDspVersion, 95
 - ASX_Tuner_GetHdRadioSdkVersion, 95
 - ASX_Tuner_GetHdRadioSignalQuality, 96
 - ASX_Tuner_GetMode, 96
 - ASX_Tuner_GetProgram, 96
 - ASX_Tuner_GetRawRFLevel, 97
 - ASX_Tuner_GetRFLevel, 97
 - ASX_Tuner_GetStatus, 97
 - ASX_Tuner_SetBand, 98
 - ASX_Tuner_SetDabAudioService, 98
 - ASX_Tuner_SetDeemphasis, 98
 - ASX_Tuner_SetFrequency, 99
 - ASX_Tuner_SetGain, 99
 - ASX_Tuner_SetHdBlend, 100
 - ASX_Tuner_SetMode, 100
 - ASX_Tuner_SetProgram, 100
- Tuner control functions, 86
- type

- asxParameterRangeInfo, [165](#)
- u
 - asxParameterRangeInfo, [165](#)
- uItems
 - asxParameterValue, [166](#)
- value
 - asxParameterRangeInfo, [165](#)
 - asxParameterRangeInfo_NamedEnumerated,
[166](#)
 - asxParameterValue, [166](#)
- Volume
 - ASX_Volume_GetChannels, [78](#)
 - ASX_Volume_GetGain, [78](#)
 - ASX_Volume_GetMute, [79](#)
 - ASX_Volume_GetRange, [79](#)
 - ASX_Volume_SetAutofade, [80](#)
 - ASX_Volume_SetGain, [80](#)
 - ASX_Volume_SetMute, [81](#)
- Volume control functions, [77](#)
- Vox
 - ASX_Vox_GetLevel, [117](#)
 - ASX_Vox_GetRange, [117](#)
 - ASX_Vox_SetLevel, [117](#)
- Vox control functions, [116](#)