

SOUND4 BIGVOICE.CL Library [1.0.0]

Generated by Doxygen 1.9.1

1 SOUND4 BIGVOICE.CL Library	1
1.1 General usage	1
1.2 Examples	1
2 Deprecated List	3
3 Module Index	5
3.1 Modules	5
4 Namespace Index	7
4.1 Namespace List	7
5 Hierarchical Index	9
5.1 Class Hierarchy	9
6 Class Index	11
6.1 Class List	11
7 File Index	13
7.1 File List	13
8 Module Documentation	15
8.1 Sample using C interface	15
8.1.1 Sample to create the processing instance	15
8.1.2 Sample processing loop	16
8.1.3 Sample to stop the processing instance	16
8.2 Environment variables and parameters	17
8.2.1 Parameters defaulting from environment	17
8.2.1.1 Webserver authentication setup :	17
8.2.1.2 Webserver secure :	17
8.2.1.3 Network access :	17
8.2.2 Parameters	17
8.2.3 Preset sharing	18
8.3 Global setup	18
8.3.1 Detailed Description	18
8.3.2 Macro Definition Documentation	19
8.3.2.1 BIGVOICE_AUDIO_INPUT_COUNT	19
8.3.2.2 BIGVOICE_AUDIO_OUTPUT_COUNT	19
8.3.2.3 BIGVOICE_AUDIOFRAME_COUNT	19
8.3.2.4 BIGVOICE_CHANNEL_COUNT	20
8.3.2.5 BIGVOICE_SAMPLE_RATE	20
8.3.3 Typedef Documentation	20
8.3.3.1 bigvoice_loggerfn	20
8.3.4 Enumeration Type Documentation	20
8.3.4.1 bigvoice_LogSeverity	20

8.3.5 Function Documentation	21
8.3.5.1 bigvoice_GetAudioInputCount()	21
8.3.5.2 bigvoice_GetAudioOutputCount()	21
8.3.5.3 bigvoice_GetChannelCount()	22
8.3.5.4 bigvoice_GetChunkSizeInFrames()	22
8.3.5.5 bigvoice_GetPossibleChunkSizeInFrames()	22
8.3.5.6 bigvoice_GetSampleRate()	23
8.3.5.7 bigvoice_GetVersion()	23
8.3.5.8 bigvoice_SetLoggerCallback()	23
8.3.5.9 bigvoice_SetLogSeverity()	24
8.4 Process creation/destruction	24
8.4.1 Detailed Description	24
8.4.2 Function Documentation	24
8.4.2.1 bigvoice_ExitProcess()	24
8.4.2.2 bigvoice_FreeParameters()	25
8.4.2.3 bigvoice_FreeParameterValue()	25
8.4.2.4 bigvoice_GetParameter()	25
8.4.2.5 bigvoice_GetProcessChunkFrames()	25
8.4.2.6 bigvoice_InitProcess()	26
8.4.2.7 bigvoice_InitProcess2()	26
8.4.2.8 bigvoice_InitProcess3()	27
8.4.2.9 bigvoice_NewParameters()	28
8.4.2.10 bigvoice_SetParameter()	28
8.4.2.11 bigvoice_TerminateProcess()	28
8.5 Instance's bus	29
8.5.1 Detailed Description	29
8.5.2 Function Documentation	29
8.5.2.1 bigvoice_FreeBus()	29
8.5.2.2 bigvoice_NewBus()	30
8.5.2.3 bigvoice_SetInstanceBus()	30
8.6 Custom preset management	30
8.6.1 Detailed Description	31
8.6.2 Typedef Documentation	31
8.6.2.1 bigvoice_storage_exists	31
8.6.2.2 bigvoice_storage_getall	32
8.6.2.3 bigvoice_storage_getall_free	32
8.6.2.4 bigvoice_storage_reader	32
8.6.2.5 bigvoice_storage_reader_free	33
8.6.2.6 bigvoice_storage_remove	33
8.6.2.7 bigvoice_storage_rename	34
8.6.2.8 bigvoice_storage_writer	34
8.6.2.9 fs_char	34

8.6.3 Enumeration Type Documentation	35
8.6.3.1 bigvoice_PresetChange_Kind	35
8.6.4 Function Documentation	35
8.6.4.1 bigvoice_PresetManager_InformChange()	35
8.6.4.2 bigvoice_SetPresetManager()	35
8.7 Metadata Management	36
8.7.1 Detailed Description	36
8.7.2 Function Documentation	36
8.7.2.1 bigvoice_FreeMetadataInfos()	36
8.7.2.2 bigvoice_GetMetadataInfos()	37
8.7.2.3 bigvoice_SetMetadata()	37
8.8 Update Management	38
8.8.1 Detailed Description	38
8.8.2 Function Documentation	38
8.8.2.1 bigvoice_StartUpdateThread()	38
8.8.2.2 bigvoice_StopUpdateThread()	38
8.8.2.3 bigvoice_WaitUpdateThreadReady()	39
8.9 Direct processing	39
8.9.1 Detailed Description	39
8.9.2 Function Documentation	39
8.9.2.1 bigvoice_GetBufferIn()	39
8.9.2.2 bigvoice_GetBufferOut()	40
8.9.2.3 bigvoice_GetEstimatedDelay()	40
8.9.2.4 bigvoice_ProcessAudio()	41
8.9.2.5 bigvoice_ProcessAudio_Planar()	41
8.10 Audio managing (any audio)	42
8.10.1 Detailed Description	42
8.10.2 Enumeration Type Documentation	42
8.10.2.1 bigvoice_SampleFormat	42
8.10.3 Function Documentation	43
8.10.3.1 bigvoice_AddAudio()	43
8.10.3.2 bigvoice_AddPadAudio()	44
8.10.3.3 bigvoice_GetAudio()	44
8.10.3.4 bigvoice_GetBytesFromFormat()	44
8.10.3.5 bigvoice_GetFormatFromName()	45
8.10.3.6 bigvoice_GetFormatName()	45
8.10.3.7 bigvoice_GetMaxPacketFrame()	45
8.10.3.8 bigvoice_GetOutputCount()	47
8.11 Audio convert helpers	47
8.11.1 Detailed Description	47
8.11.2 Function Documentation	47
8.11.2.1 bigvoice_AudioConvertFrom()	48

8.11.2.2 bigvoice_AudioConvertTo()	48
8.11.2.3 bigvoice_AudioMonoFromLiveStereo()	48
8.11.2.4 bigvoice_AudioMonoToLiveStereo()	49
8.11.2.5 bigvoice_MonoToStereo()	49
8.11.2.6 bigvoice_MonoToStereo_Planar()	50
8.11.2.7 bigvoice_StereoToMono()	50
8.11.2.8 bigvoice_StereoToMono_Planar()	50
8.12 Client's JSon config interfacing	51
8.12.1 Detailed Description	51
8.12.2 Function Documentation	51
8.12.2.1 bigvoice_DeleteClient()	51
8.12.2.2 bigvoice_FreeJsonAnswer()	52
8.12.2.3 bigvoice_NewClient()	52
8.12.2.4 bigvoice_ProcessJson()	52
8.12.2.5 bigvoice_SaveState()	53
8.13 JSON syntax	53
8.13.1 Commands	54
8.13.2 Answers	54
8.13.3 Examples	55
8.13.3.1 Get status	55
8.13.3.2 Get all changes since last request	55
8.13.3.3 Check license	55
8.13.3.4 Set Processor name	55
8.13.3.5 Get presetlist	55
8.13.3.6 Load a preset	55
8.13.3.7 Get information on preset on air	55
8.13.3.8 Set input gain	56
8.13.3.9 Set bypass	56
8.13.3.10 Export a preset	56
8.13.3.11 Import a preset	56
8.14 JSON syntax for internal setup	56
8.14.1 Get process and modules information	57
8.14.2 Authentication management	57
8.14.2.1 Difference between setup and admin account	57
8.14.2.2 Access encoding	57
8.14.2.3 Change setup account	57
8.14.2.4 Get user info	58
8.14.2.5 Add or modify a user	58
8.14.2.6 Removes a user	58
8.14.2.7 List users	58
8.14.2.8 Get/set unlogged access	58
8.14.2.9 Get realm	58

8.14.2.10 Get API version	59
8.14.2.11 Get setup is admin	59
8.14.2.12 Get/set setup account	59
8.14.2.13 Get/set custom storage	59
8.14.3 Get process' interface description	59
8.14.3.1 Get XML decription	59
8.14.3.2 Get Process Image	59
8.14.3.3 Get Process Ico	60
8.15 Web server	60
8.15.1 Detailed Description	60
8.15.2 Macro Definition Documentation	60
8.15.2.1 SOUND4_INVALID_WEBSERVER_ID	60
8.15.2.2 SOUND4_WEBSERVER_HTTP_OK	61
8.15.2.3 SOUND4_WEBSERVER_HTTPS_OK	61
8.15.3 Function Documentation	61
8.15.3.1 bigvoice_Webserver()	61
8.15.3.2 bigvoice_Webserver_Status()	62
8.15.3.3 bigvoice_Webserver_Stop()	62
8.15.3.4 bigvoice_Webserver_tcp()	63
8.15.3.5 bigvoice_Webserver_tcp2()	63
8.16 Sample using C++ interface wrapper	64
8.16.1 Setup and sanity checks	64
8.16.2 Sample to create the processing instance	64
8.16.3 Communicating with the instance	64
8.16.4 Starting a web-server	64
8.16.5 Sample processing loop	65
8.17 Sample using C++ interface wrapper with dynamic library loading	65
8.17.1 Dynamic loading	65
8.17.2 Setup and sanity checks	65
8.17.3 Sample to create the processing instance	65
8.17.4 Communicating with the instance	65
8.17.5 Starting a web-server	65
8.17.6 Sample processing loop	66
8.17.7 Custom preset manager	66
8.17.8 Custom preset manager loading	67
9 Namespace Documentation	69
9.1 dyn Namespace Reference	69
9.1.1 Detailed Description	69
9.2 sound4 Namespace Reference	69
9.2.1 Detailed Description	69
9.3 sound4::bigvoice Namespace Reference	69

9.3.1 Detailed Description	70
9.3.2 Typedef Documentation	70
9.3.2.1 log_cb_t	70
9.3.2.2 LogSeverity	70
9.3.2.3 PresetChange_Kind	71
9.3.2.4 SampleFormat	71
9.3.3 Variable Documentation	71
9.3.3.1 ChunkFrames	71
9.3.3.2 InputChannels	71
9.3.3.3 InputSampleSize	71
9.3.3.4 OutputChannels	72
9.3.3.5 OutputSampleSize	72
9.3.3.6 SampleRate	72
9.4 sound4::bigvoice::dyn Namespace Reference	72
9.4.1 Typedef Documentation	73
9.4.1.1 log_cb_t	73
9.4.1.2 LogSeverity	73
9.4.1.3 PresetChange_Kind	73
9.4.1.4 SampleFormat	73
9.4.2 Variable Documentation	74
9.4.2.1 ChunkFrames	74
9.4.2.2 InputChannels	74
9.4.2.3 InputSampleSize	74
9.4.2.4 OutputChannels	74
9.4.2.5 OutputSampleSize	74
9.4.2.6 SampleRate	75
9.5 sound4::bigvoice::dyn::helper Namespace Reference	75
9.5.1 Detailed Description	75
9.5.2 Typedef Documentation	75
9.5.2.1 DynFunc_t	75
9.5.2.2 DynLib_t	76
9.6 sound4::bigvoice::helper Namespace Reference	76
9.6.1 Detailed Description	76
9.7 std Namespace Reference	76
10 Class Documentation	77
10.1 bigvoice_CBus Struct Reference	77
10.1.1 Detailed Description	77
10.2 bigvoice_CClientInstance Struct Reference	77
10.2.1 Detailed Description	77
10.3 bigvoice_CInstance Struct Reference	77
10.3.1 Detailed Description	78

10.4 bigvoice_CParameters Struct Reference	78
10.4.1 Detailed Description	78
10.5 sound4::bigvoice::CBus Class Reference	78
10.5.1 Detailed Description	78
10.5.2 Constructor & Destructor Documentation	78
10.5.2.1 CBus()	79
10.5.2.2 ~CBus()	79
10.5.3 Member Function Documentation	79
10.5.3.1 Get()	79
10.6 sound4::bigvoice::dyn::CBus Class Reference	79
10.6.1 Detailed Description	79
10.6.2 Constructor & Destructor Documentation	80
10.6.2.1 CBus()	80
10.6.2.2 ~CBus()	80
10.6.3 Member Function Documentation	80
10.6.3.1 Get()	80
10.7 sound4::bigvoice::CInstance::CClient Class Reference	80
10.7.1 Detailed Description	81
10.7.2 Constructor & Destructor Documentation	81
10.7.2.1 CClient() [1/2]	81
10.7.2.2 CClient() [2/2]	81
10.7.2.3 ~CClient()	81
10.7.3 Member Function Documentation	81
10.7.3.1 operator=()	81
10.7.3.2 ProcessJson()	81
10.8 sound4::bigvoice::dyn::CInstance::CClient Class Reference	82
10.8.1 Detailed Description	82
10.8.2 Constructor & Destructor Documentation	83
10.8.2.1 CClient() [1/2]	83
10.8.2.2 CClient() [2/2]	83
10.8.2.3 ~CClient()	83
10.8.3 Member Function Documentation	83
10.8.3.1 operator=()	83
10.8.3.2 ProcessJson()	83
10.9 sound4::bigvoice::dyn::CDynLib Class Reference	84
10.9.1 Detailed Description	86
10.9.2 Constructor & Destructor Documentation	86
10.9.2.1 CDynLib()	86
10.9.3 Member Function Documentation	86
10.9.3.1 IsOk()	86
10.9.3.2 Load()	86
10.9.4 Member Data Documentation	87

10.9.4.1 AddAudio	87
10.9.4.2 AddPadAudio	87
10.9.4.3 AudioConvertFrom	87
10.9.4.4 AudioConvertTo	87
10.9.4.5 AudioMonoFromLiveStereo	87
10.9.4.6 AudioMonoToLiveStereo	88
10.9.4.7 DeleteClient	88
10.9.4.8 ExitProcess	88
10.9.4.9 FreeBus	88
10.9.4.10 FreeJsonAnswer	88
10.9.4.11 FreeMetadataInfos	88
10.9.4.12 FreeParameters	89
10.9.4.13 FreeParameterValue	89
10.9.4.14 GetAudio	89
10.9.4.15 GetAudioInputCount	89
10.9.4.16 GetAudioOutputCount	89
10.9.4.17 GetBufferIn	89
10.9.4.18 GetBufferOut	90
10.9.4.19 GetBytesFromFormat	90
10.9.4.20 GetChannelCount	90
10.9.4.21 GetChunkSizeInFrames	90
10.9.4.22 GetEstimatedDelay	90
10.9.4.23 GetFormatFromName	90
10.9.4.24 GetFormatName	91
10.9.4.25 GetMaxPacketFrame	91
10.9.4.26 GetMetadataInfos	91
10.9.4.27 GetOutputCount	91
10.9.4.28 GetParameter	91
10.9.4.29 GetPossibleChunkSizeInFrames	91
10.9.4.30 GetProcessChunkFrames	92
10.9.4.31 GetSampleRate	92
10.9.4.32 GetVersion	92
10.9.4.33 InitProcess	92
10.9.4.34 InitProcess2	92
10.9.4.35 InitProcess3	92
10.9.4.36 MonoToStereo	93
10.9.4.37 MonoToStereo_Planar	93
10.9.4.38 NewBus	93
10.9.4.39 NewClient	93
10.9.4.40 NewParameters	93
10.9.4.41 PresetManager_InformChange	93
10.9.4.42 ProcessAudio	94

10.9.4.43 ProcessAudio_Planar	94
10.9.4.44 ProcessJson	94
10.9.4.45 SaveState	94
10.9.4.46 SetInstanceBus	94
10.9.4.47 SetLoggerCallback	94
10.9.4.48 SetLogSeverity	95
10.9.4.49 SetMetadata	95
10.9.4.50 SetParameter	95
10.9.4.51 SetPresetManager	95
10.9.4.52 StartUpdateThread	95
10.9.4.53 StereoToMono	95
10.9.4.54 StereoToMono_Planar	96
10.9.4.55 StopUpdateThread	96
10.9.4.56 TerminateProcess	96
10.9.4.57 WaitUpdateThreadReady	96
10.9.4.58 Webserver	96
10.9.4.59 Webserver_Status	96
10.9.4.60 Webserver_Stop	97
10.9.4.61 Webserver_tcp	97
10.9.4.62 Webserver_tcp2	97
10.10 sound4::bigvoice::dyn::helper::CDynLoader Class Reference	97
10.10.1 Detailed Description	98
10.10.2 Constructor & Destructor Documentation	98
10.10.2.1 CDynLoader() [1/3]	98
10.10.2.2 CDynLoader() [2/3]	98
10.10.2.3 CDynLoader() [3/3]	98
10.10.2.4 ~CDynLoader()	98
10.10.3 Member Function Documentation	98
10.10.3.1 Close()	99
10.10.3.2 GetSymbol()	99
10.10.3.3 GetSymbol_safe()	99
10.10.3.4 GetThisLibraryPath()	99
10.10.3.5 IsOk()	99
10.10.3.6 Load()	100
10.10.3.7 operator bool()	100
10.10.3.8 operator=()	100
10.11 sound4::bigvoice::CInstance Class Reference	100
10.11.1 Detailed Description	101
10.11.2 Constructor & Destructor Documentation	101
10.11.2.1 CInstance() [1/2]	102
10.11.2.2 CInstance() [2/2]	102
10.11.2.3 ~CInstance()	102

10.11.3 Member Function Documentation	102
10.11.3.1 AddAudio()	102
10.11.3.2 Create()	102
10.11.3.3 GetAudio()	103
10.11.3.4 GetBufferIn()	103
10.11.3.5 GetBufferOut()	104
10.11.3.6 GetChunkFrames()	104
10.11.3.7 GetEstimatedDelay()	104
10.11.3.8 GetMetadataInfos()	105
10.11.3.9 GetParam()	105
10.11.3.10 IsOk()	105
10.11.3.11 NewClient()	105
10.11.3.12 OnUpdateThreadStart()	106
10.11.3.13 OnUpdateThreadStop()	106
10.11.3.14 operator=()	106
10.11.3.15 PresetManager_InformChange()	106
10.11.3.16 ProcessAnyAudio()	106
10.11.3.17 ProcessAudio()	107
10.11.3.18 ProcessAudio_Planar()	107
10.11.3.19 SetBus()	108
10.11.3.20 SetMetadata()	108
10.11.3.21 SetParam()	108
10.11.3.22 SetPresetManager()	109
10.11.3.23 StartWebServer()	109
10.11.3.24 Stop()	110
10.11.3.25 StopWebServer()	110
10.11.3.26 Terminate()	110
10.12 sound4::bigvoice::dyn::CInstance Class Reference	110
10.12.1 Detailed Description	111
10.12.2 Constructor & Destructor Documentation	112
10.12.2.1 CInstance() [1/2]	112
10.12.2.2 CInstance() [2/2]	112
10.12.2.3 ~CInstance()	112
10.12.3 Member Function Documentation	112
10.12.3.1 AddAudio()	112
10.12.3.2 Create()	112
10.12.3.3 GetAudio()	113
10.12.3.4 GetBufferIn()	113
10.12.3.5 GetBufferOut()	114
10.12.3.6 GetChunkFrames()	114
10.12.3.7 GetEstimatedDelay()	114
10.12.3.8 GetMetadataInfos()	115

10.12.3.9	GetParam()	115
10.12.3.10	IsOk()	115
10.12.3.11	NewClient()	115
10.12.3.12	OnUpdateThreadStart()	116
10.12.3.13	OnUpdateThreadStop()	116
10.12.3.14	operator=()	116
10.12.3.15	PresetManager_InformChange()	116
10.12.3.16	ProcessAnyAudio()	116
10.12.3.17	ProcessAudio()	117
10.12.3.18	ProcessAudio_Planar()	117
10.12.3.19	SetBus()	118
10.12.3.20	SetMetadata()	118
10.12.3.21	SetParam()	118
10.12.3.22	SetPresetManager()	119
10.12.3.23	StartWebServer()	119
10.12.3.24	Stop()	120
10.12.3.25	StopWebServer()	120
10.12.3.26	Terminate()	120
10.13	sound4::bigvoice::CPresetLoader Class Reference	120
10.13.1	Detailed Description	121
10.13.2	Constructor & Destructor Documentation	121
10.13.2.1	CPresetLoader()	121
10.13.2.2	~CPresetLoader()	121
10.13.3	Member Function Documentation	121
10.13.3.1	Exists()	121
10.13.3.2	GetAll()	122
10.13.3.3	IsReadOnly()	122
10.13.3.4	Read()	122
10.13.3.5	Remove()	122
10.13.3.6	Rename()	122
10.13.3.7	Write()	122
10.14	sound4::bigvoice::dyn::CPresetLoader Class Reference	123
10.14.1	Detailed Description	123
10.14.2	Constructor & Destructor Documentation	123
10.14.2.1	CPresetLoader()	123
10.14.2.2	~CPresetLoader()	123
10.14.3	Member Function Documentation	123
10.14.3.1	Exists()	124
10.14.3.2	GetAll()	124
10.14.3.3	IsReadOnly()	124
10.14.3.4	Read()	124
10.14.3.5	Remove()	124

10.14.3.6 Rename()	124
10.14.3.7 Write()	125
10.15 sound4::bigvoice::dyn::helper::DynFuncHolder< T > Class Template Reference	125
10.15.1 Detailed Description	125
10.15.2 Constructor & Destructor Documentation	125
10.15.2.1 DynFuncHolder() [1/2]	125
10.15.2.2 DynFuncHolder() [2/2]	126
10.15.3 Member Function Documentation	126
10.15.3.1 IsOk()	126
10.15.3.2 operator bool()	126
10.15.3.3 operator T*()	126
10.16 MyPresetManager Class Reference	126
10.16.1 Detailed Description	127
10.16.2 Constructor & Destructor Documentation	127
10.16.2.1 MyPresetManager()	127
10.16.2.2 ~MyPresetManager()	127
10.16.3 Member Function Documentation	127
10.16.3.1 Exists()	127
10.16.3.2 GetAll()	128
10.16.3.3 IsReadOnly()	128
10.16.3.4 Read()	128
10.16.3.5 Remove()	128
10.16.3.6 Rename()	128
10.16.3.7 SetupPath()	129
10.16.3.8 Write()	129
10.17 sound4::bigvoice::dyn::helper::SampleFormat< T > Struct Template Reference	129
10.17.1 Detailed Description	129
10.18 sound4::bigvoice::helper::SampleFormat< T > Struct Template Reference	130
10.18.1 Detailed Description	130
10.19 sound4::bigvoice::dyn::helper::SampleFormat< float > Struct Reference	130
10.19.1 Detailed Description	130
10.19.2 Member Data Documentation	130
10.19.2.1 format	131
10.20 sound4::bigvoice::helper::SampleFormat< float > Struct Reference	131
10.20.1 Detailed Description	131
10.20.2 Member Data Documentation	131
10.20.2.1 format	131
10.21 sound4::bigvoice::dyn::helper::SampleFormat< int16_t > Struct Reference	131
10.21.1 Detailed Description	132
10.21.2 Member Data Documentation	132
10.21.2.1 format	132
10.22 sound4::bigvoice::helper::SampleFormat< int16_t > Struct Reference	132

10.22.1 Detailed Description	132
10.22.2 Member Data Documentation	132
10.22.2.1 format	133
10.23 sound4::bigvoice::dyn::helper::SampleFormat< int32_t > Struct Reference	133
10.23.1 Detailed Description	133
10.23.2 Member Data Documentation	133
10.23.2.1 format	133
10.24 sound4::bigvoice::helper::SampleFormat< int32_t > Struct Reference	133
10.24.1 Detailed Description	134
10.24.2 Member Data Documentation	134
10.24.2.1 format	134
11 File Documentation	135
11.1 documentation.h File Reference	135
11.2 documentation.h	135
11.3 sound4.bigvoice.cl-pipe-cxx.cpp File Reference	135
11.3.1 Function Documentation	135
11.3.1.1 GetEnv()	136
11.3.1.2 main()	136
11.3.1.3 MyLogger()	136
11.3.1.4 ProcessLoop()	136
11.3.1.5 Usage()	137
11.4 sound4.bigvoice.cl-pipe-cxx.cpp	137
11.5 sound4.bigvoice.cl-pipe-dyn.cpp File Reference	139
11.5.1 Function Documentation	139
11.5.1.1 GetEnv()	139
11.5.1.2 main()	140
11.5.1.3 MyLogger()	140
11.5.1.4 ProcessLoop()	140
11.5.1.5 Usage()	141
11.6 sound4.bigvoice.cl-pipe-dyn.cpp	141
11.7 sound4.bigvoice.cl-pipe.cpp File Reference	144
11.7.1 Macro Definition Documentation	144
11.7.1.1 CHANNEL_COUNT	145
11.7.1.2 CHUNK_SIZE	145
11.7.1.3 SAMPLE_RATE	145
11.7.2 Function Documentation	145
11.7.2.1 main()	145
11.7.2.2 MyLogger()	146
11.7.2.3 ProcessLoop()	146
11.7.2.4 Usage()	146
11.8 sound4.bigvoice.cl-pipe.cpp	146

11.9 sound4.bigvoice.cl.h File Reference	149
11.9.1 Macro Definition Documentation	152
11.9.1.1 BIGVOICE_DECL	152
11.9.1.2 BIGVOICE_HAS_CLOUDBUS	152
11.9.1.3 BIGVOICE_HAS_WEBSERVER	152
11.10 sound4.bigvoice.cl.h	153
11.11 sound4.bigvoice.cl.hpp File Reference	156
11.11.1 Macro Definition Documentation	158
11.11.1.1 fs_strdup	158
11.12 sound4.bigvoice.cl.hpp	158
11.13 sound4.bigvoice.cl_dyn.hpp File Reference	165
11.13.1 Detailed Description	166
11.13.2 Macro Definition Documentation	166
11.13.2.1 fs_strdup	166
11.14 sound4.bigvoice.cl_dyn.hpp	167
Index	181

Chapter 1

SOUND4 BIGVOICE.CL Library

1.1 General usage

You may check library configuration and setup logging [Global setup](#) and [Environment variables and parameters](#)

You have to create/destroy an instance [Process creation/destruction](#).

For audio, you can either use direct optimized form with [Direct processing](#) or you can use [Audio managing \(any audio\)](#) to have any format. You may use [Audio convert helpers](#).

You need to start a thread calling [bigvoice_StartUpdateThread](#) see [Update Management](#)

You can start a web server for configuration, see [Web server](#)

If it is needed, you can use metadata : see [Metadata Management](#)

If you need, you may use your own custom preset manager for saving files and presets. See [Custom preset management](#)

To access the presets/settings, you can either use JSON socket from [Update Management](#) or do all manually with [Client's JSon config interfacing](#) using [JSON syntax](#)

See full example [cpp/sound4.bigvoice.cl-pipe.cpp](#)

You can also use the C++ wrapper from header [sound4.bigvoice.cl.hpp](#) or the one with dynamic library loading [sound4.bigvoice.cl_dyn.hpp](#).

Note

In all library, all `char*` are **UTF-8** encoded

1.2 Examples

See [Sample using C interface](#)

See [Sample using C++ interface wrapper](#)

See [Sample using C++ interface wrapper with dynamic library loading](#)

Chapter 2

Deprecated List

Member [bigvoice_GetChunkSizeInFrames](#) ()

Use [bigvoice_GetPossibleChunkSizeInFrames](#)

Member [bigvoice_InitProcess](#) (const char *LoginKey, const char *RadioName, const char *AWS_Access_ID, const char *AWS_Access_Secret, const char *save_path)

Use [bigvoice_InitProcess3](#)

Member [bigvoice_InitProcess2](#) (const char *LoginKey, const char *RadioName, const char *AWS_Access_ID, const char *AWS_Access_Secret, const char *save_path, const struct [bigvoice_CParameters](#) *parameters)

Use [bigvoice_InitProcess3](#)

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Sample using C interface	15
Environment variables and parameters	17
Global setup	18
Process creation/destruction	24
Instance's bus	29
Custom preset managagement	30
Metadata Management	36
Update Management	38
Direct processing	39
Audio managing (any audio)	42
Audio convert helpers	47
Client's JSON config interfacing	51
JSON syntax	53
JSON syntax for internal setup	56
Web server	60
Sample using C++ interface wrapper	64
Sample using C++ interface wrapper with dynamic library loading	65

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

dyn	69
sound4	69
sound4::bigvoice	69
sound4::bigvoice::dyn	72
sound4::bigvoice::dyn::helper	75
sound4::bigvoice::helper	76
std	76

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

bigvoice_CBus	77
bigvoice_CClientInstance	77
bigvoice_CInstance	77
bigvoice_CParameters	78
sound4::bigvoice::CBus	78
sound4::bigvoice::dyn::CBus	79
sound4::bigvoice::CInstance::CClient	80
sound4::bigvoice::dyn::CInstance::CClient	82
sound4::bigvoice::dyn::CDynLib	84
sound4::bigvoice::dyn::helper::CDynLoader	97
sound4::bigvoice::CInstance	100
sound4::bigvoice::dyn::CInstance	110
sound4::bigvoice::CPresetLoader	120
sound4::bigvoice::dyn::CPresetLoader	123
MyPresetManager	126
sound4::bigvoice::dyn::helper::DynFuncHolder< T >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_AddAudio) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_AddPadAudio) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_AudioConvertFrom) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_AudioConvertTo) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_AudioMonoFromLiveStereo) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_AudioMonoToLiveStereo) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_DeleteClient) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_ExitProcess) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_FreeBus) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_FreeJsonAnswer) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_FreeMetadataInfos) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_FreeParameters) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_FreeParameterValue) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetAudio) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetAudioInputCount) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetAudioOutputCount) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetBufferIn) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetBufferOut) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetBytesFromFormat) >	125

sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetChannelCount) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetChunkSizeInFrames) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetEstimatedDelay) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetFormatFromName) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetFormatName) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetMaxPacketFrame) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetMetadataInfos) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetOutputCount) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetParameter) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetPossibleChunkSizeInFrames) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetProcessChunkFrames) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetSampleRate) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_GetVersion) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_InitProcess) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_InitProcess2) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_InitProcess3) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_MonoToStereo) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_MonoToStereo_Planar) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_NewBus) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_NewClient) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_NewParameters) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_PresetManager_InformChange) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_ProcessAudio) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_ProcessAudio_Planar) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_ProcessJson) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_SaveState) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_SetInstanceBus) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_SetLoggerCallback) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_SetLogSeverity) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_SetMetadata) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_SetParameter) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_SetPresetManager) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_StartUpdateThread) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_StereoToMono) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_StereoToMono_Planar) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_StopUpdateThread) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_TerminateProcess) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_WaitUpdateThreadReady) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_Webserver) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_Webserver_Status) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_Webserver_Stop) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_Webserver_tcp) >	125
sound4::bigvoice::dyn::helper::DynFuncHolder< decltype(bigvoice_Webserver_tcp2) >	125
sound4::bigvoice::dyn::helper::SampleFormat< T >	129
sound4::bigvoice::helper::SampleFormat< T >	130
sound4::bigvoice::dyn::helper::SampleFormat< float >	130
sound4::bigvoice::helper::SampleFormat< float >	131
sound4::bigvoice::dyn::helper::SampleFormat< int16_t >	131
sound4::bigvoice::helper::SampleFormat< int16_t >	132
sound4::bigvoice::dyn::helper::SampleFormat< int32_t >	133
sound4::bigvoice::helper::SampleFormat< int32_t >	133

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

bigvoice_CBus	77
bigvoice_CClientInstance	77
bigvoice_CInstance	77
bigvoice_CParameters	78
sound4::bigvoice::CBus	78
sound4::bigvoice::dyn::CBus	79
sound4::bigvoice::CInstance::CClient	80
sound4::bigvoice::dyn::CInstance::CClient	82
sound4::bigvoice::dyn::CDynLib	
Dynamic library interface	84
sound4::bigvoice::dyn::helper::CDynLoader	
Helper to load dynamic library	97
sound4::bigvoice::CInstance	100
sound4::bigvoice::dyn::CInstance	110
sound4::bigvoice::CPresetLoader	120
sound4::bigvoice::dyn::CPresetLoader	123
sound4::bigvoice::dyn::helper::DynFuncHolder< T >	
Helper to wrap a function pointer	125
MyPresetManager	
[Custom preset manager]	126
sound4::bigvoice::dyn::helper::SampleFormat< T >	
Helper for Sample format types	129
sound4::bigvoice::helper::SampleFormat< T >	
Helper for Sample format types	130
sound4::bigvoice::dyn::helper::SampleFormat< float >	130
sound4::bigvoice::helper::SampleFormat< float >	131
sound4::bigvoice::dyn::helper::SampleFormat< int16_t >	131
sound4::bigvoice::helper::SampleFormat< int16_t >	132
sound4::bigvoice::dyn::helper::SampleFormat< int32_t >	133
sound4::bigvoice::helper::SampleFormat< int32_t >	133

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

documentation.h	135
sound4.bigvoice.cl-pipe-cxx.cpp	
C++ sample using C++ interface wrapper	135
sound4.bigvoice.cl-pipe-dyn.cpp	
C++ sample using C++ interface wrapper with dynamic library loading	139
sound4.bigvoice.cl-pipe.cpp	
C++ sample using only C interface	144
sound4.bigvoice.cl.h	
C interface for library	149
sound4.bigvoice.cl.hpp	
C++ wrapper for library	156
sound4.bigvoice.cl_dyn.hpp	
C++ wrapper for dynamic loading library	165

Chapter 8

Module Documentation

8.1 Sample using C interface

See full example [cpp/sound4.bigvoice.cl-pipe.cpp](#)

8.1.1 Sample to create the processing instance

see [Process creation/destruction](#) and [Update Management](#)

```
// Setup parameters
struct bigvoice_CParameters *params = bigvoice_NewParameters();
// Optional: activate web authentication, set admin password
// This creates the authentication only for a new storage path
bigvoice_SetParameter(params, "ADMIN_USER", "admin");
bigvoice_SetParameter(params, "ADMIN_SECRET", "admin");

// Create the processing instance
pInstance=bigvoice_InitProcess2(pLoginKey, pRadioName, pKeyId, pAccessKey, store_path.c_str(), params);
bigvoice_FreeParameters(params);
if (!pInstance) {
    std::cerr << "Failed to create the process instance" << std::endl;
    return 1;
}
// Start the update thread, without JSON server (low priority)
std::thread updater([pInstance]() {
    // will returned when stopped or destroyed
    std::cerr << "bigvoice_StartUpdateThread (no json port)" << std::endl;
    bigvoice_StartUpdateThread(pInstance, 0);
    std::cerr << "bigvoice_StartUpdateThread END" << std::endl;
});
// This is not mandatory, but better be sure we are ready right now
std::cerr << "Waiting update thread" << std::endl;
if (bigvoice_WaitUpdateThreadReady(pInstance, 5000)<0) {
    std::cerr << "Timeout to create the process instance" << std::endl;
}
std::cerr << "Waiting update thread DONE" << std::endl;
#if (BIGVOICE_HAS_WEBSERVER)
// Starts a web server, listening on port http://localhost:8080, https://localhost:8443
std::cerr << "Starting web server" << std::endl;
int listenport= 8080;
int listenport_secure= 8443;
uint64_t webserver= SOUND4_INVALID_WEBSERVER_ID;
if (listenport || listenport_secure) {
    webserver = bigvoice_Webserver(listenport, listenport_secure, pInstance);
    if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
        std::cerr << "Starting web server FAILED" << std::endl;
    } else {
        int web_status=bigvoice_Webserver_Status(webserver);
        if (listenport) {
            if (web_status & SOUND4_WEBSERVER_HTTP_OK) {
                std::cerr << "Web server listening on port " << listenport;
            } else {
                std::cerr << "Web server failed to listen on port " << listenport;
            }
        }
    }
}
```

```

        if (listenport_secure) {
            if (web_status & SOUND4_WEBSERVER_HTTPS_OK) {
                std::cerr << "Web server listening on secure port " << listenport_secure;
            } else {
                std::cerr << "Web server failed to listen on secure port " << listenport_secure;
            }
        }
        std::cerr << "Starting web server DONE" << std::endl;
    }
}
#endif // (BIGVOICE_HAS_WEBSERVER)

// Force load a factory preset now
// Could be used instead of setting a store_path
/*{
    bigvoice_CClientInstance *pClient=bigvoice_NewClient(pInstance);
    const char *answer=bigvoice_ProcessJson(pClient, "{ \"factoryload\": \"All Programs\\\" }", NULL);
    std::cerr << "Loaded factory preset answer: " << answer<< std::endl;
    bigvoice_FreeJsonAnswer(answer);
    bigvoice_DeleteClient(pClient);
}*/

```

8.1.2 Sample processing loop

see [Direct processing](#)

```

bool stopthread=false;
alignas(64) float audio_in[CHUNK_SIZE*CHANNEL_COUNT];
alignas(64) float audio_out[CHUNK_SIZE*CHANNEL_COUNT];
#ifdef _WIN32 || defined(WIN32)
    // Windows: The stdin, stdout, and stderr streams always open in text mode by default
    _setmode(_fileno(stdin), _O_BINARY);
    _setmode(_fileno(stdout), _O_BINARY);
#endif

while (!stopthread)
{
    // Read a chunk from stdin
    if (std::fread(&audio_in[0], sizeof(float), CHUNK_SIZE*CHANNEL_COUNT, stdin) !=
        CHUNK_SIZE*CHANNEL_COUNT) {
        std::cerr << "ProcessLoop: input stopped" << std::endl;
        stopthread=true;
        break;
    }
    if (stopthread) break;
    // Process it
    bigvoice_ProcessAudio(pInstance, audio_in, audio_out);
    // Write it to stdout
    if (fwrite(&audio_out[0], sizeof(float), CHUNK_SIZE*CHANNEL_COUNT, stdout) !=
        CHUNK_SIZE*CHANNEL_COUNT) {
        std::cerr << "Failed to write audio output" << std::endl;
        stopthread=true;
    }
}

```

8.1.3 Sample to stop the processing instance

see [Process creation/destruction](#)

```

#ifdef (BIGVOICE_HAS_WEBSERVER)
    // Stopping the webserver
    std::cerr << "Stopping web server" << std::endl;
    bigvoice_Webserver_Stop(webserver, 1000);
    std::cerr << "Stopping web server DONE" << std::endl;
#endif // (BIGVOICE_HAS_WEBSERVER)

// Stopping the processing instance
std::cerr << "Stopping Update thread" << std::endl;
bigvoice_StopUpdateThread(pInstance); // this will exit the update thread
std::cerr << "Stopping Update thread DONE" << std::endl;
updater.join();
bigvoice_ExitProcess(pInstance);

```

8.2 Environment variables and parameters

8.2.1 Parameters defaulting from environment

The following environment variables are used by software library:

Note

Those variables can be passed as parameters with [bigvoice_InitProcess2](#)

8.2.1.1 Webserver authentication setup :

- `SETUP_USER` see [bigvoice_InitProcess](#)
- `SETUP_SECRET` see [bigvoice_InitProcess](#)
- `ADMIN_USER` see [bigvoice_InitProcess](#)
- `ADMIN_SECRET` see [bigvoice_InitProcess](#)

8.2.1.2 Webserver secure :

- `SSL_CERTIFICATE_PATH` : path to the SSL public certificate
- `SSL_PRIVATE_PATH` : path to the SSL private certificate

8.2.1.3 Network access :

- `LICENSE_SERVER` : the SOUND4 license server's URL
- `LICENSE_SERVER_PROXY` : proxy server to use for licenses [user:pass@]host:port
- `LICENSE_CERT_DIR` : optional certificates dirs for HTTPS verification
- `LICENSE_CERT_CHECK` : if "false", do not check HTTPS certificates

8.2.2 Parameters

The following parameters can be set before creating instance, but are never taken from environment.

Run mode:

- `FILE_MODE=seconds` : force file mode, settings the maximum estimated audio duration (non realtime, wait license ok to process)
- `STATE_FILENAME=filename` : force this instance to store its state in file named `filename.json` (instead of `state.json`)
- `STATE_ONAIR=filename` : force this instance to store its on air preset name in file named `filename.onair` (instead of none). Used for sharing between instances.
- `STORAGE_RO=notempty` : force the storage in read-only mode
- `STATE_RO=notempty` : force the state in read-only mode (but can save presets/auth)
- `WAIT_LICENSE=notempty` : force to wait the license load before starting (always in file mode)

8.2.3 Preset sharing

You can use the same folder for multiple instances so they share their presets.

In this case, you should use a different `STATE_FILENAME` per instance so they keep their settings and on air presets separated. If not, each time settings or onair preset are changed, the changes will be reflected to the other instances using same `STATE_FILENAME`.

You may use the same `STATE_ONAIR` for multiple instances sharing the folder, so that changing the on air preset on one immediately changes the on air preset on other instances using the same `STATE_ONAIR`. If the instance has pending unsaved changes in the on air preset, it will not be updated automatically to avoid losing changes.

8.3 Global setup

Macros

- `#define BIGVOICE_AUDIOFRAME_COUNT 12`
- `#define BIGVOICE_CHANNEL_COUNT 1`
- `#define BIGVOICE_AUDIO_INPUT_COUNT 1`
- `#define BIGVOICE_AUDIO_OUTPUT_COUNT 1`
- `#define BIGVOICE_SAMPLE_RATE 48000`

Typedefs

- `typedef void(* bigvoice_loggerfn) (enum bigvoice_LogSeverity, const char *)`

Enumerations

- `enum bigvoice_LogSeverity {
 none = 0 , fatal = 1 , error = 2 , warning = 3 ,
 info = 4 , verbose = 5 , verbose2 = 6 , verbose3 = 7 ,
 verbose4 = 8 , verbose5 = 9 }`

Functions

- `const char * bigvoice_GetVersion ()`
- `unsigned int bigvoice_GetChunkSizeInFrames ()`
- `unsigned int * bigvoice_GetPossibleChunkSizeInFrames ()`
- `unsigned int bigvoice_GetChannelCount ()`
- `unsigned int bigvoice_GetAudioInputCount ()`
- `unsigned int bigvoice_GetAudioOutputCount ()`
- `unsigned int bigvoice_GetSampleRate ()`
- `void bigvoice_SetLoggerCallback (bigvoice_loggerfn logger)`
- `void bigvoice_SetLogSeverity (enum bigvoice_LogSeverity severity)`

8.3.1 Detailed Description

Get information on the library. Set logging mode

8.3.2 Macro Definition Documentation

8.3.2.1 BIGVOICE_AUDIO_INPUT_COUNT

```
#define BIGVOICE_AUDIO_INPUT_COUNT 1
```

This version number of inputs for this process.

Remarks

For binary compatibility, this should be runtime checked to be the same as [bigvoice_GetAudioInputCount\(\)](#)

Definition at line 166 of file [sound4.bigvoice.cl.h](#).

8.3.2.2 BIGVOICE_AUDIO_OUTPUT_COUNT

```
#define BIGVOICE_AUDIO_OUTPUT_COUNT 1
```

This version number of outputs for this process.

Remarks

For binary compatibility, this should be runtime checked to be the same as [bigvoice_GetAudioOutputCount\(\)](#)

Definition at line 172 of file [sound4.bigvoice.cl.h](#).

8.3.2.3 BIGVOICE_AUDIOFRAME_COUNT

```
#define BIGVOICE_AUDIOFRAME_COUNT 12
```

This version number of frames in a process chunk.

Remarks

For binary compatibility, this should be runtime checked to be the same as [bigvoice_GetChunkSizeInFrames\(\)](#)

Definition at line 154 of file [sound4.bigvoice.cl.h](#).

8.3.2.4 BIGVOICE_CHANNEL_COUNT

```
#define BIGVOICE_CHANNEL_COUNT 1
```

This version number of channels.

Remarks

For binary compatibility, this should be runtime checked to be the same as [bigvoice_GetChannelCount\(\)](#)

Definition at line 160 of file [sound4.bigvoice.cl.h](#).

8.3.2.5 BIGVOICE_SAMPLE_RATE

```
#define BIGVOICE_SAMPLE_RATE 48000
```

This input/output sample-rate.

Remarks

For binary compatibility, this should be runtime checked to be the same as [bigvoice_GetSampleRate\(\)](#)

Definition at line 178 of file [sound4.bigvoice.cl.h](#).

8.3.3 Typedef Documentation

8.3.3.1 bigvoice_loggerfn

```
typedef void(* bigvoice_loggerfn) (enum bigvoice_LogSeverity, const char *)
```

The logging callback format

Warning

Do not block/wait in the callback, it can be called from any place in code

Note

In all library, all `char*` are **UTF-8** encoded

Definition at line 271 of file [sound4.bigvoice.cl.h](#).

8.3.4 Enumeration Type Documentation

8.3.4.1 bigvoice_LogSeverity

```
enum bigvoice_LogSeverity
```

The logging levels

Enumerator

none	
fatal	
error	
warning	
info	
verbose	
verbose2	
verbose3	
verbose4	
verbose5	

Definition at line 252 of file [sound4.bigvoice.cl.h](#).

8.3.5 Function Documentation

8.3.5.1 `bigvoice_GetAudioInputCount()`

```
unsigned int bigvoice_GetAudioInputCount ( )
```

Get the number of inputs for this process

Remarks

All inputs have [bigvoice_GetChannelCount\(\)](#) channels
Fixed to 1, but should be checked for backward compatibility

Returns

The process input count

8.3.5.2 `bigvoice_GetAudioOutputCount()`

```
unsigned int bigvoice_GetAudioOutputCount ( )
```

Get the number of outputs for this process

Remarks

All outputs have [bigvoice_GetChannelCount\(\)](#) channels
Fixed to 1, but should be checked for backward compatibility

Returns

The process outputs count

8.3.5.3 bigvoice_GetChannelCount()

```
unsigned int bigvoice_GetChannelCount ( )
```

Get the number of channels needed by this process

Remarks

Fixed to 1, but should be checked for backward compatibility

Returns

The process channels count

8.3.5.4 bigvoice_GetChunkSizeInFrames()

```
unsigned int bigvoice_GetChunkSizeInFrames ( )
```

Get the number of frames in a process chunk.

The output will be available only by chunks of this size.

Note

A Frame is a set of [bigvoice_GetChannelCount](#) samples

Remarks

Fixed to 12, but should be checked for backward compatibility

Returns

The process chunk size in frame

Deprecated Use [bigvoice_GetPossibleChunkSizeInFrames](#)

8.3.5.5 bigvoice_GetPossibleChunkSizeInFrames()

```
unsigned int* bigvoice_GetPossibleChunkSizeInFrames ( )
```

Get the possibles number of frames in a process chunk.

The output will be available only by chunks of this size.

Note

A Frame is a set of [bigvoice_GetChannelCount](#) samples

Returns

An array of possible process chunk size in frame, ending with 0

8.3.5.6 bigvoice_GetSampleRate()

```
unsigned int bigvoice_GetSampleRate ( )
```

Get the processing sample-rate

Remarks

For now, always return 48000, but should be checked for backward compatibility

Returns

The process sample-rate in Hz

8.3.5.7 bigvoice_GetVersion()

```
const char* bigvoice_GetVersion ( )
```

Get the process version string

Returns

version string

8.3.5.8 bigvoice_SetLoggerCallback()

```
void bigvoice_SetLoggerCallback (
    bigvoice_loggerfn logger )
```

Set a callback for logging

If not set, a default logger prints to stderr.

Warning

Do not block/wait in the callback, it can be called from any place in code

Parameters

<i>logger</i>	A C callback called for logging.
---------------	----------------------------------

8.3.5.9 bigvoice_SetLogSeverity()

```
void bigvoice_SetLogSeverity (
    enum bigvoice_LogSeverity severity )
```

Change the global log severity

Logs above this level will not be printed.

Parameters

<i>severity</i>	Log severity
-----------------	--------------

8.4 Process creation/destruction

Classes

- struct [bigvoice_CInstance](#)
- struct [bigvoice_CParameters](#)

Functions

- struct [bigvoice_CParameters](#) * [bigvoice_NewParameters](#) ()
- void [bigvoice_FreeParameters](#) (struct [bigvoice_CParameters](#) *params)
- void [bigvoice_SetParameter](#) (struct [bigvoice_CParameters](#) *params, const char *name, const char *value)
- const char * [bigvoice_GetParameter](#) (struct [bigvoice_CParameters](#) *params, const char *name)
- void [bigvoice_FreeParameterValue](#) (const char *value)
- struct [bigvoice_CInstance](#) * [bigvoice_InitProcess](#) (const char *LoginKey, const char *RadioName, const char *AWS_Access_ID, const char *AWS_Access_Secret, const char *save_path)
- struct [bigvoice_CInstance](#) * [bigvoice_InitProcess2](#) (const char *LoginKey, const char *RadioName, const char *AWS_Access_ID, const char *AWS_Access_Secret, const char *save_path, const struct [bigvoice_CParameters](#) *parameters)
- struct [bigvoice_CInstance](#) * [bigvoice_InitProcess3](#) (const char *LoginKey, const char *RadioName, const char *AWS_Access_ID, const char *AWS_Access_Secret, const char *save_path, const struct [bigvoice_CParameters](#) *parameters, unsigned int frames_per_chunk)
- unsigned int [bigvoice_GetProcessChunkFrames](#) (struct [bigvoice_CInstance](#) *instance)
- int [bigvoice_TerminateProcess](#) (struct [bigvoice_CInstance](#) *instance)
- void [bigvoice_ExitProcess](#) (struct [bigvoice_CInstance](#) *instance)

8.4.1 Detailed Description

Create and destroy a process instance, and retrieve some general information about the process.

8.4.2 Function Documentation

8.4.2.1 bigvoice_ExitProcess()

```
void bigvoice_ExitProcess (
    struct bigvoice_CInstance * instance )
```

Quit and destroy the instance

Parameters

<i>instance</i>	the instance to destroy
-----------------	-------------------------

8.4.2.2 bigvoice_FreeParameters()

```
void bigvoice_FreeParameters (
    struct bigvoice_CParameters * params )
```

Free the parameters created with [bigvoice_NewParameters](#).

8.4.2.3 bigvoice_FreeParameterValue()

```
void bigvoice_FreeParameterValue (
    const char * value )
```

Free a parameter value returned by [bigvoice_GetParameter](#)

Parameters

<i>value</i>	the returned value
--------------	--------------------

8.4.2.4 bigvoice_GetParameter()

```
const char* bigvoice_GetParameter (
    struct bigvoice_CParameters * params,
    const char * name )
```

Get current parameter value.

Returned value must be freed with [bigvoice_FreeParameterValue](#)

8.4.2.5 bigvoice_GetProcessChunkFrames()

```
unsigned int bigvoice_GetProcessChunkFrames (
    struct bigvoice_CInstance * instance )
```

Get the number of audio frames in each chunk of processing.

The value reflects the frames_per_chunk parameter passed to [bigvoice_InitProcess3](#)

Parameters

<i>instance</i>	the instance to destroy
-----------------	-------------------------

Returns

the number of audio frames in each chunk of processing

Note

A Frame is a set of [bigvoice_GetChannelCount](#) samples

8.4.2.6 bigvoice_InitProcess()

```
struct bigvoice\_CInstance* bigvoice_InitProcess (
    const char * LoginKey,
    const char * RadioName,
    const char * AWS_Access_ID,
    const char * AWS_Access_Secret,
    const char * save_path )
```

Create a process instance with default parameters (from env)

Same as [bigvoice_InitProcess2](#), but with default parameters (from env)

Deprecated Use [bigvoice_InitProcess3](#)

8.4.2.7 bigvoice_InitProcess2()

```
struct bigvoice\_CInstance* bigvoice_InitProcess2 (
    const char * LoginKey,
    const char * RadioName,
    const char * AWS_Access_ID,
    const char * AWS_Access_Secret,
    const char * save_path,
    const struct bigvoice\_CParameters * parameters )
```

Create a process instance with default frames_per_chunk (12)

Same as [bigvoice_InitProcess3](#), but with default frames_per_chunk

Deprecated Use [bigvoice_InitProcess3](#)

8.4.2.8 bigvoice_InitProcess3()

```
struct bigvoice_CInstance* bigvoice_InitProcess3 (
    const char * LoginKey,
    const char * RadioName,
    const char * AWS_Access_ID,
    const char * AWS_Access_Secret,
    const char * save_path,
    const struct bigvoice_CParameters * parameters,
    unsigned int frames_per_chunk )
```

Create a process instance

The output will have to be freed with `bigvoice_ExitProcess`

Attention

The **save_path** must be an existing folder with R/W access, or be NULL (then it uses volatile memory storage)

If the `save_path` is NULL or the folder is empty, a default configuration with a default preset will be loaded, unless using [Custom preset management](#).

If used in a container, we recommend to put a volume for this path, so it will survive updates.

Web login initialization

If the storage is not yet initialized, you can choose how to protect the web access.
There are 2 modes:

1. Setup mode: you define a setup account, which can then create an administrator account.
The setup account always keeps the access and can delete the administrator.
2. Admin mode: you define the administrator, which will be the single administrator.

To select the mode, you have to put in the parameters

1. `SETUP_USER` (and optionally `SETUP_SECRET`) to select Setup mode
2. `ADMIN_USER` (and optionally `ADMIN_SECRET`) to select Admin mode
3. none: the web access is opened, no login will be asked

If you do not set the `xxx_SECRET`, then the setuppassword will be randomly generated and print in logs (at info severity).

You can also use the JSON api to change this after the storage is created : see [Change setup account](#)

License access

The licenses are retrieved via Internet and access some servers. You may use an alternate server by changing `LICENSE_SERVER` in the parameters.

Parameters

<i>LoginKey</i>	SOUND4 Login key (was User ID)
<i>RadioName</i>	SOUND4 RadioName
<i>AWS_Access_ID</i>	SOUND4 AWS access ID
<i>AWS_Access_Secret</i>	SOUND4 AWS access Secret
<i>save_path</i>	Path where we can load/save the configuration. If NULL, memory storage will be used.
<i>parameters</i>	Parameters
<i>frames_per_chunk</i>	The number of audio frames in each chunk of processing. Must be in the list returned by bigvoice_GetPossibleChunkSizeInFrames

Returns

A new process instance

8.4.2.9 bigvoice_NewParameters()

```
struct bigvoice_CParameters* bigvoice_NewParameters ( )
```

Create parameters to use with [bigvoice_InitProcess2](#).

The output will have to be freed with [bigvoice_FreeParameters](#)

The default values are taken from environment (for compatibility).

8.4.2.10 bigvoice_SetParameter()

```
void bigvoice_SetParameter (
    struct bigvoice_CParameters * params,
    const char * name,
    const char * value )
```

Set a new parameter.

Some values are in taken from env, so see [Environment variables and parameters](#) for usable variables

See also [Custom preset managagement](#)

See also [Instance's bus](#)

8.4.2.11 bigvoice_TerminateProcess()

```
int bigvoice_TerminateProcess (
    struct bigvoice_CInstance * instance )
```

Terminate the processing

This is useful only with file processing, to get the final status. For some plugins, this is mandatory

Parameters

<i>instance</i>	the instance to terminate
-----------------	---------------------------

Returns

0 is success, <0 if error, >0 if some status

8.5 Instance's bus

This allows instances to exchange real-time data.

Classes

- struct [bigvoice_CBus](#)

Functions

- struct [bigvoice_CBus](#) * [bigvoice_NewBus](#) ()
- void [bigvoice_FreeBus](#) (struct [bigvoice_CBus](#) *bus)
- void [bigvoice_SetInstanceBus](#) (struct [bigvoice_CParameters](#) *params, struct [bigvoice_CBus](#) *bus)

8.5.1 Detailed Description

The bus is used to manage Dominate-it between instances.

8.5.2 Function Documentation

8.5.2.1 [bigvoice_FreeBus\(\)](#)

```
void bigvoice_FreeBus (  
    struct bigvoice\_CBus * bus )
```

Frees the bus allocated with [bigvoice_NewBus](#)

Parameters

<i>bus</i>	The bus to free
------------	-----------------

8.5.2.2 `bigvoice_NewBus()`

```
struct bigvoice\_CBus* bigvoice_NewBus ( )
```

Creates a new bus

Returns

the new bus, to be freed with [bigvoice_FreeBus](#)

8.5.2.3 `bigvoice_SetInstanceBus()`

```
void bigvoice_SetInstanceBus (
    struct bigvoice\_CParameters * params,
    struct bigvoice\_CBus * bus )
```

Set the instance's bus. It can be used by instances to exchange real-time information if needed.

Attention

The bus should be created with the same library

Parameters

<i>params</i>	the parameters to configure
<i>bus</i>	the bus to use

8.6 Custom preset management

This allows to provide a customized preset management.

Typedefs

- typedef wchar_t [fs_char](#)
- typedef char *(* [bigvoice_storage_reader](#)) (const [fs_char](#) *filename, void *userdata)
- typedef void(* [bigvoice_storage_reader_free](#)) (char *content, void *userdata)
- typedef int(* [bigvoice_storage_writer](#)) (const [fs_char](#) *filename, const char *content, void *userdata)
- typedef int(* [bigvoice_storage_exists](#)) (const [fs_char](#) *filename, void *userdata)
- typedef [fs_char](#) *(* [bigvoice_storage_getall](#)) (void *userdata)
- typedef void(* [bigvoice_storage_getall_free](#)) ([fs_char](#) **all, void *userdata)
- typedef int(* [bigvoice_storage_remove](#)) (const [fs_char](#) *filename, void *userdata)
- typedef int(* [bigvoice_storage_rename](#)) (const [fs_char](#) *from, const [fs_char](#) *to, void *userdata)

Enumerations

- enum `bigvoice_PresetChange_Kind` { `change_kind_created` = 1, `change_kind_modified`, `change_kind_deleted` }

Functions

- void `bigvoice_SetPresetManager` (struct `bigvoice_CParameters` *params, `bigvoice_storage_reader` reader, `bigvoice_storage_reader_free`, `bigvoice_storage_writer` writer, `bigvoice_storage_exists` exists, `bigvoice_storage_getall` getall, `bigvoice_storage_getall_free` getall_free, `bigvoice_storage_remove` remove, `bigvoice_storage_rename` rename, int `IsReadOnly`, void *userdata)
- void `bigvoice_PresetManager_InformChange` (struct `bigvoice_CInstance` *instance, const `fs_char` *relative_path, enum `bigvoice_PresetChange_Kind` change_kind)

8.6.1 Detailed Description

This allows to provide a customized storage for presets and settings.

By default, the library store data on disk or in memory (see `save_path` parameter in `bigvoice_InitProcess2`), but you may use these functions to store data.

It can be used for remote storage, database storage, ...

Known files:

- Files with name ending with `.pst` are presets
- `state.json` is the instance state file, holding settings and unsaved preset changes.
- `auth.json` is the storage for web authentication settings.
- `*.onair` are used for sharing preset on air name between instances (see `STATE_ONAIR` in [Parameters](#)).

Attention

`state.json` name can be overridden per instance with `STATE_FILENAME` in parameters, see [Parameters](#).

File names are in OS native characters (`wchar_t` for Windows, `char` in **UTF-8** for others)

8.6.2 Typedef Documentation

8.6.2.1 `bigvoice_storage_exists`

```
typedef int(* bigvoice_storage_exists) (const fs_char *filename, void *userdata)
```

Function to call to test if a stored file exists.

Parameters

<i>filename</i>	the name of the file to test.
<i>userdata</i>	the userdata passed to bigvoice_SetPresetManager

Returns

-1 if not exists, 0 if it exists

Definition at line 578 of file [sound4.bigvoice.cl.h](#).

8.6.2.2 bigvoice_storage_getall

```
typedef fs_char**(* bigvoice_storage_getall) (void *userdata)
```

Function to call to get all stored files' name. Result has to be freed with [bigvoice_storage_getall_free](#) callback

Parameters

<i>userdata</i>	the userdata passed to bigvoice_SetPresetManager
-----------------	--

Returns

A NULL terminated array of file names

Definition at line 587 of file [sound4.bigvoice.cl.h](#).

8.6.2.3 bigvoice_storage_getall_free

```
typedef void(* bigvoice_storage_getall_free) (fs_char **all, void *userdata)
```

Function to call to free the result of getall.

Parameters

<i>all</i>	the returned value from bigvoice_storage_getall
<i>userdata</i>	the userdata passed to bigvoice_SetPresetManager

Definition at line 595 of file [sound4.bigvoice.cl.h](#).

8.6.2.4 bigvoice_storage_reader

```
typedef char*(* bigvoice_storage_reader) (const fs_char *filename, void *userdata)
```

Function to call to read a stored file, returning its content.

Parameters

<i>filename</i>	the name of the file to read.
<i>userdata</i>	the userdata passed to bigvoice_SetPresetManager

Returns

the **UTF-8** content of the file

Definition at line 551 of file [sound4.bigvoice.cl.h](#).

8.6.2.5 bigvoice_storage_reader_free

```
typedef void(* bigvoice_storage_reader_free) (char *content, void *userdata)
```

Function to call to free the result of `bigvoice_storage_reader`.

Parameters

<i>content</i>	the returned value from <code>bigvoice_storage_reader</code>
<i>userdata</i>	the userdata passed to bigvoice_SetPresetManager

Definition at line 559 of file [sound4.bigvoice.cl.h](#).

8.6.2.6 bigvoice_storage_remove

```
typedef int(* bigvoice_storage_remove) (const fs_char *filename, void *userdata)
```

Function to call to remove a stored file.

Parameters

<i>filename</i>	the name of the file to remove.
<i>userdata</i>	the userdata passed to bigvoice_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 604 of file [sound4.bigvoice.cl.h](#).

8.6.2.7 bigvoice_storage_rename

```
typedef int(* bigvoice_storage_rename) (const fs_char *from, const fs_char *to, void *userdata)
```

Function to call to rename a stored file.

Parameters

<i>from</i>	The source file name (must exist)
<i>to</i>	The destination file name (must not exist)
<i>userdata</i>	the userdata passed to bigvoice_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 614 of file [sound4.bigvoice.cl.h](#).

8.6.2.8 bigvoice_storage_writer

```
typedef int(* bigvoice_storage_writer) (const fs_char *filename, const char *content, void *userdata)
```

Function to call to write a stored file.

Parameters

<i>filename</i>	the name of the file to write.
<i>content</i>	the UTF-8 content to write to the file
<i>userdata</i>	the userdata passed to bigvoice_SetPresetManager

Returns

-1 if failed, 0 if OK

Definition at line 569 of file [sound4.bigvoice.cl.h](#).

8.6.2.9 fs_char

```
typedef wchar_t fs_char
```

Definition at line 539 of file [sound4.bigvoice.cl.h](#).

8.6.3 Enumeration Type Documentation

8.6.3.1 `bigvoice_PresetChange_Kind`

enum `bigvoice_PresetChange_Kind`

Kind of change on a storage path

Enumerator

<code>change_kind_created</code>	Path has been created.
<code>change_kind_modified</code>	Path has been modified.
<code>change_kind_deleted</code>	Path has been deleted.

Definition at line 647 of file [sound4.bigvoice.cl.h](#).

8.6.4 Function Documentation

8.6.4.1 `bigvoice_PresetManager_InformChange()`

```
void bigvoice_PresetManager_InformChange (
    struct bigvoice_CInstance * instance,
    const fs_char * relative_path,
    enum bigvoice_PresetChange_Kind change_kind )
```

Inform that a change happened in the storage. This must be used only in conjunction with [bigvoice_SetPresetManager](#)

Parameters

<i>instance</i>	the processing instance
<i>relative_path</i>	The path which changed, relative to the root storage
<i>change_kind</i>	What happened to this path

8.6.4.2 `bigvoice_SetPresetManager()`

```
void bigvoice_SetPresetManager (
    struct bigvoice_CParameters * params,
    bigvoice_storage_reader reader,
    bigvoice_storage_reader_free ,
    bigvoice_storage_writer writer,
```

```

bigvoice_storage_exists exists,
bigvoice_storage_getall getall,
bigvoice_storage_getall_free getall_free,
bigvoice_storage_remove remove,
bigvoice_storage_rename rename,
int IsReadOnly,
void * userdata )

```

Setup a custom preset manager. This overrides the `save_path` in [bigvoice_InitProcess2](#)

To dynamically update when storage change, see [bigvoice_PresetManager_InformChange](#)

Parameters

<i>params</i>	The params to set up
<i>reader</i>	The function to read a file
<i>writer</i>	The function to write a file
<i>exists</i>	The function to test for a file presence
<i>getall</i>	The function to get all files in storage
<i>getall_free</i>	The function to free the result from <i>getall</i>
<i>remove</i>	The function to remove a file from storage
<i>rename</i>	The function to rename a file
<i>IsReadOnly</i>	Set to 1 if storage is read-only
<i>userdata</i>	The userdata parameter for callbacks

8.7 Metadata Management

Functions

- void [bigvoice_SetMetadata](#) (struct [bigvoice_CInstance](#) *instance, const char *key, const char *value)
- const char ** [bigvoice_GetMetadataInfos](#) (struct [bigvoice_CInstance](#) *instance)
- void [bigvoice_FreeMetadataInfos](#) (struct [bigvoice_CInstance](#) *instance, const char **infos)

8.7.1 Detailed Description

Metadata handling.

Metadata is used on-need, and might be used by extensions modules (Watermarking, Streaming, ...)

8.7.2 Function Documentation

8.7.2.1 bigvoice_FreeMetadataInfos()

```

void bigvoice_FreeMetadataInfos (
    struct bigvoice\_CInstance * instance,
    const char ** infos )

```

Frees the information returned by [bigvoice_GetMetadataInfos](#).

Parameters

<i>instance</i>	the instance
<i>infos</i>	the infos

8.7.2.2 bigvoice_GetMetadataInfos()

```
const char** bigvoice_GetMetadataInfos (  
    struct bigvoice\_CInstance * instance )
```

Get information on known metadata keys.

Parameters

<i>instance</i>	the instance
-----------------	--------------

Returns

a list of key,descriptions, null terminated, of known metadata keys used in process and extensions. May be null. Should be freed with [bigvoice_FreeMetadataInfos](#)

8.7.2.3 bigvoice_SetMetadata()

```
void bigvoice_SetMetadata (  
    struct bigvoice\_CInstance * instance,  
    const char * key,  
    const char * value )
```

Adds the specific metadata to the process.

It can come from any thread, and will be used as soon as possible. Setting a null value erase the metadata, which is different from setting an empty string. Initial metadata might be set as soon as instance is created (and before update thread).

Parameters

<i>instance</i>	the instance
<i>key</i>	the metadata key. If key contains a dot (like wm.KEY), it will be sent only to the matching plugin (ie wm here) without the prefix (ie KEY here). Use empty prefix for routing to main only.
<i>value</i>	the metadata value or nullptr to erase it.

8.8 Update Management

Functions

- void [bigvoice_StartUpdateThread](#) (struct [bigvoice_CInstance](#) *instance, unsigned int port)
- void [bigvoice_StopUpdateThread](#) (struct [bigvoice_CInstance](#) *instance)
- int [bigvoice_WaitUpdateThreadReady](#) (struct [bigvoice_CInstance](#) *instance, int milliseconds)

8.8.1 Detailed Description

Manage the process update.

8.8.2 Function Documentation

8.8.2.1 [bigvoice_StartUpdateThread\(\)](#)

```
void bigvoice_StartUpdateThread (
    struct bigvoice\_CInstance * instance,
    unsigned int port )
```

Update function

This needs to run in a thread, which can have low priority. It will also allow the JSON access via TCP.

Remarks

This function does not return until it is stopped with [bigvoice_StopUpdateThread](#)

Parameters

<i>instance</i>	the instance
<i>port</i>	the port to listen for incoming JSON connections. If 0, the JSON server is not accessible

8.8.2.2 [bigvoice_StopUpdateThread\(\)](#)

```
void bigvoice_StopUpdateThread (
    struct bigvoice\_CInstance * instance )
```

Stop the update thread

Parameters

<i>instance</i>	the processing instance
-----------------	-------------------------

8.8.2.3 bigvoice_WaitUpdateThreadReady()

```
int bigvoice_WaitUpdateThreadReady (
    struct bigvoice_CInstance * instance,
    int milliseconds )
```

Wait for the update thread to be ready

Parameters

<i>instance</i>	the processing instance
<i>milliseconds</i>	maximum time to wait in milliseconds

Returns

-1 if timeout waiting for the thread, 0 if OK

8.9 Direct processing

Functions

- void [bigvoice_ProcessAudio](#) (struct [bigvoice_CInstance](#) *instance, const float *input, float *output)
- void [bigvoice_ProcessAudio_Planar](#) (struct [bigvoice_CInstance](#) *instance, float const *const *input, float *const *output)
- float * [bigvoice_GetBufferIn](#) (struct [bigvoice_CInstance](#) *instance)
- float * [bigvoice_GetBufferOut](#) (struct [bigvoice_CInstance](#) *instance)
- unsigned int [bigvoice_GetEstimatedDelay](#) (struct [bigvoice_CInstance](#) *instance)

8.9.1 Detailed Description

Optimized processing with direct in->out.

8.9.2 Function Documentation

8.9.2.1 bigvoice_GetBufferIn()

```
float* bigvoice_GetBufferIn (
    struct bigvoice_CInstance * instance )
```

Get preallocated aligned buffer for audio input for [bigvoice_ProcessAudio](#)()

The buffer is a float[256 * [bigvoice_GetAudioInputCount](#)() * [bigvoice_GetChannelCount](#)()]

So for this process/version : float[256*1*1]

Note

The calling process can allocate the buffer itself, but it must be aligned.

Parameters

<i>instance</i>	the processing instance
-----------------	-------------------------

Returns

An audio input dedicated buffer.

8.9.2.2 bigvoice_GetBufferOut()

```
float* bigvoice_GetBufferOut (
    struct bigvoice_CInstance * instance )
```

Get preallocated aligned buffer for audio output for [bigvoice_ProcessAudio\(\)](#)

The buffer is a float[256 * [bigvoice_GetAudioOutputCount\(\)](#) * [bigvoice_GetChannelCount\(\)](#)]

So for this process/version : float[256*1*1]

Note

The calling process can allocate the buffer itself, but it must be aligned.

Parameters

<i>instance</i>	the processing instance
-----------------	-------------------------

Returns

An audio output dedicated buffer.

8.9.2.3 bigvoice_GetEstimatedDelay()

```
unsigned int bigvoice_GetEstimatedDelay (
    struct bigvoice_CInstance * instance )
```

Get the delay (estimated) of current processing algorithm

Remarks

This is not very accurate, it does not take into account group delays.

Returns

Estimated delay in frames

8.9.2.4 `bigvoice_ProcessAudio()`

```
void bigvoice_ProcessAudio (
    struct bigvoice\_CInstance * instance,
    const float * input,
    float * output )
```

Process an audio chunk

A chunk contains [bigvoice_GetProcessChunkFrames](#) frames, set at init by `frames_per_chunk` in [bigvoice_InitProcess3](#)

Input has [bigvoice_GetAudioInputCount\(\)](#) * [bigvoice_GetChannelCount\(\)](#) channels

Output has [bigvoice_GetAudioOutputCount\(\)](#) * [bigvoice_GetChannelCount\(\)](#) channels

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are interlaced, so the buffers are [L0,R0,L1,R1,...,Ln,Rn] (L=Left, R=Right)

In case of multiple audio input/output, buffer is [aL0,aR0,bL0,bR0,...,aLn,aRn,bLn,bRn]

Warning

Buffers must be 64-byte aligned

Note

With 64 frames per chunk and 2 channels and 1 audio input, input buffer size is 64*1*2*4=512 bytes

Parameters

<i>instance</i>	the processing instance
<i>input</i>	audio input buffer (64 byte aligned) or NULL to use internal buffer bigvoice_GetBufferIn
<i>output</i>	audio output buffer (64 byte aligned) or NULL to use internal buffer bigvoice_GetBufferOut

8.9.2.5 `bigvoice_ProcessAudio_Planar()`

```
void bigvoice_ProcessAudio_Planar (
    struct bigvoice\_CInstance * instance,
    float const *const * input,
    float *const * output )
```

Process an audio chunk of planar audio

A chunk contains [bigvoice_GetProcessChunkFrames](#) frames, set at init by `frames_per_chunk` in [bigvoice_InitProcess3](#)

Input has [bigvoice_GetAudioInputCount\(\)](#) * [bigvoice_GetChannelCount\(\)](#) channels

Output has [bigvoice_GetAudioOutputCount\(\)](#) * [bigvoice_GetChannelCount\(\)](#) channels

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are NOT interlaced, so the buffers are arrays of planar audio : [[L0,L1,...,Ln] , [R0,R1,...,Rn]] (L=Left, R=Right)

In case of multiple audio input/output, buffer are [[aL...], [aR...], [bL...], [bR...], ...]

Parameters

<i>instance</i>	the processing instance
<i>input</i>	array of audio input buffers for each input channel (no alignment required)
<i>output</i>	array of audio output buffers for each input channel (no alignment required)

8.10 Audio managing (any audio)

Enumerations

- enum `bigvoice_SampleFormat` {
`INVALID_FORMAT` = 0, `S16_LE`, `S16_BE`, `S24_LE`,
`S24_BE`, `S32_LE`, `S32_BE`, `F32_LE`,
`F32_BE`, `S16_NATIVE` = `S16_LE`, `S24_NATIVE` = `S24_LE`, `S32_NATIVE` = `S32_LE`,
`F32_NATIVE` = `F32_LE` }

Functions

- const char * `bigvoice_GetFormatName` (const enum `bigvoice_SampleFormat` fmt)
- enum `bigvoice_SampleFormat` `bigvoice_GetFormatFromName` (const char *name)
- unsigned int `bigvoice_GetBytesFromFormat` (const enum `bigvoice_SampleFormat` fmt)
- unsigned int `bigvoice_GetMaxPacketFrame` (struct `bigvoice_CInstance` *instance)
- unsigned int `bigvoice_AddAudio` (struct `bigvoice_CInstance` *instance, const uint8_t *payload, unsigned int nFrame, enum `bigvoice_SampleFormat` fmt)
- unsigned int `bigvoice_AddPadAudio` (struct `bigvoice_CInstance` *instance)
- unsigned int `bigvoice_GetOutputCount` (struct `bigvoice_CInstance` *instance)
- unsigned int `bigvoice_GetAudio` (struct `bigvoice_CInstance` *instance, uint8_t *payload, unsigned int max↔_nFrame, enum `bigvoice_SampleFormat` fmt)

8.10.1 Detailed Description

This allows various payload size and formats, at the expense of internal conversions. This can be useful when receiving audio from network.

Remarks

It is more optimized to use [Direct processing](#).

8.10.2 Enumeration Type Documentation

8.10.2.1 bigvoice_SampleFormat

```
enum bigvoice_SampleFormat
```

Supported formats for audio I/O

Enumerator

INVALID_FORMAT	Invalid.
S16_LE	16-bit signed integer, little-endian
S16_BE	16-bit signed integer, big-endian
S24_LE	24-bit signed integer, little-endian
S24_BE	24-bit signed integer, big-endian
S32_LE	32-bit signed integer, little-endian
S32_BE	32-bit signed integer, big-endian
F32_LE	32-bit floating-point, little-endian
F32_BE	32-bit floating-point, big-endian
S16_NATIVE	16-bit signed integer, native
S24_NATIVE	24-bit signed integer, native
S32_NATIVE	32-bit signed integer, native
F32_NATIVE	32-bit floating-point, native

Definition at line 855 of file [sound4.bigvoice.cl.h](#).

8.10.3 Function Documentation

8.10.3.1 bigvoice_AddAudio()

```
unsigned int bigvoice_AddAudio (
    struct bigvoice_CInstance * instance,
    const uint8_t * payload,
    unsigned int nFrame,
    enum bigvoice_SampleFormat fmt )
```

Convert and process the payload.

Parameters

<i>instance</i>	the processing instance
<i>payload</i>	the binary raw audio buffer
<i>nFrame</i>	the number of frames in the payload
<i>fmt</i>	the format of the payload

Returns

number of unprocessed frames (if can not store output) if non zero, consume output and restart where it stopped

8.10.3.2 bigvoice_AddPadAudio()

```
unsigned int bigvoice_AddPadAudio (
    struct bigvoice_CInstance * instance )
```

Add silence to finish the current audio chunk and process it

Parameters

<i>instance</i>	the processing instance
-----------------	-------------------------

Returns

number of added silence frame

8.10.3.3 bigvoice_GetAudio()

```
unsigned int bigvoice_GetAudio (
    struct bigvoice_CInstance * instance,
    uint8_t * payload,
    unsigned int max_nFrame,
    enum bigvoice_SampleFormat fmt )
```

Consume the output

Parameters

<i>instance</i>	the processing instance
<i>payload</i>	the binary raw audio buffer to store output
<i>max_nFrame</i>	the maximum number of frames that can be stored in the payload
<i>fmt</i>	the wanted format of the payload

Returns

number of frames put in the buffer

8.10.3.4 bigvoice_GetBytesFromFormat()

```
unsigned int bigvoice_GetBytesFromFormat (
    const enum bigvoice_SampleFormat fmt )
```

Get the format byte size

Parameters

<i>fmt</i>	the format
------------	------------

Returns

the number of bytes used by this format for a single channel audio sample

8.10.3.5 bigvoice_GetFormatFromName()

```
enum bigvoice_SampleFormat bigvoice_GetFormatFromName (
    const char * name )
```

Convert string to format

Parameters

<i>name</i>	the format string
-------------	-------------------

Returns

the matching format, or INVALID_FORMAT if invalid.

8.10.3.6 bigvoice_GetFormatName()

```
const char* bigvoice_GetFormatName (
    const enum bigvoice_SampleFormat fmt )
```

Convert format to string

Parameters

<i>fmt</i>	the format
------------	------------

Returns

the string name of the format

8.10.3.7 bigvoice_GetMaxPacketFrame()

```
unsigned int bigvoice_GetMaxPacketFrame (
    struct bigvoice_CInstance * instance )
```

Maximum packet size (in frames) that can be queued at output.

Parameters

<i>instance</i>	The process instance
-----------------	----------------------

Returns

Maximum packet size (in frames) that can be queued at output.

8.10.3.8 **bigvoice_GetOutputCount()**

```
unsigned int bigvoice_GetOutputCount (
    struct bigvoice\_CInstance * instance )
```

Get the available output size

Parameters

<i>instance</i>	the processing instance
-----------------	-------------------------

Returns

number of available frames at output

8.11 Audio convert helpers

Functions

- void [bigvoice_AudioConvertFrom](#) (const uint8_t *payload, float *output, size_t nSpl, enum [bigvoice_SampleFormat](#) fmt)
- void [bigvoice_AudioConvertTo](#) (const float *input, uint8_t *payload, size_t nSpl, enum [bigvoice_SampleFormat](#) fmt)
- void [bigvoice_StereoToMono](#) (const float *input, float *output, size_t nFrame)
- void [bigvoice_MonoToStereo](#) (const float *input, float *output, size_t nFrame)
- void [bigvoice_StereoToMono_Planar](#) (const float *inputL, const float *inputR, float *output, size_t nFrame)
- void [bigvoice_MonoToStereo_Planar](#) (const float *input, float *outputL, float *outputR, size_t nFrame)
- void [bigvoice_AudioMonoFromLiveStereo](#) (const uint8_t *payload, float *output)
- void [bigvoice_AudioMonoToLiveStereo](#) (const float *input, uint8_t *payload)

8.11.1 Detailed Description

8.11.2 Function Documentation

8.11.2.1 bigvoice_AudioConvertFrom()

```
void bigvoice_AudioConvertFrom (
    const uint8_t * payload,
    float * output,
    size_t nSpl,
    enum bigvoice_SampleFormat fmt )
```

Convert the payload to floating-point array

Warning

Output buffer must be 64-byte aligned

Parameters

<i>payload</i>	the binary raw audio buffer
<i>output</i>	audio output buffer (64 byte aligned)
<i>nSpl</i>	the number of samples (not frames)
<i>fmt</i>	the format of the payload

8.11.2.2 bigvoice_AudioConvertTo()

```
void bigvoice_AudioConvertTo (
    const float * input,
    uint8_t * payload,
    size_t nSpl,
    enum bigvoice_SampleFormat fmt )
```

Convert floating-point array to payload

Warning

Input buffer must be 64-byte aligned

Parameters

<i>input</i>	audio input buffer (64 byte aligned)
<i>payload</i>	the binary raw audio buffer
<i>nSpl</i>	the number of samples (not frames)
<i>fmt</i>	the format of the payload

8.11.2.3 bigvoice_AudioMonoFromLiveStereo()

```
void bigvoice_AudioMonoFromLiveStereo (
```

```
const uint8_t * payload,
float * output )
```

Convert Livewire Livestereo payload to floating-point Mono (12 samples)

Parameters

<i>payload</i>	Livewire Live Stereo raw buffer
<i>output</i>	audio output mono buffer (64 byte aligned)

8.11.2.4 bigvoice_AudioMonoToLiveStereo()

```
void bigvoice_AudioMonoToLiveStereo (
    const float * input,
    uint8_t * payload )
```

Convert floating-point Mono (12 samples) to Livewire Livestereo payload

Parameters

<i>input</i>	audio input mono buffer (64 byte aligned)
<i>payload</i>	Livewire Live Stereo raw buffer

8.11.2.5 bigvoice_MonoToStereo()

```
void bigvoice_MonoToStereo (
    const float * input,
    float * output,
    size_t nFrame )
```

Convert floating-point Mono to Stereo (L,L)

Warning

Buffers must be 64-byte aligned

Parameters

<i>input</i>	audio input mono buffer (64 byte aligned)
<i>output</i>	audio output stereo buffer (64 byte aligned)
<i>nFrame</i>	the number of frames

8.11.2.6 bigvoice_MonoToStereo_Planar()

```
void bigvoice_MonoToStereo_Planar (
    const float * input,
    float * outputL,
    float * outputR,
    size_t nFrame )
```

Convert floating-point Mono to Stereo (planar) (L,L)

Warning

Buffers must be 64-byte aligned

Parameters

<i>input</i>	audio input mono buffer (64 byte aligned)
<i>outputL</i>	audio output left buffer (64 byte aligned)
<i>outputR</i>	audio output right buffer (64 byte aligned)
<i>nFrame</i>	the number of frames

8.11.2.7 bigvoice_StereoToMono()

```
void bigvoice_StereoToMono (
    const float * input,
    float * output,
    size_t nFrame )
```

Convert floating-point Stereo to Mono (L+R)/2

Warning

Buffers must be 64-byte aligned

Parameters

<i>input</i>	audio input stereo buffer (64 byte aligned)
<i>output</i>	audio output mono buffer (64 byte aligned)
<i>nFrame</i>	the number of frames

8.11.2.8 bigvoice_StereoToMono_Planar()

```
void bigvoice_StereoToMono_Planar (
    const float * inputL,
```

```
const float * inputR,
float * output,
size_t nFrame )
```

Convert floating-point Stereo (planar) to Mono (L+R)/2

Warning

Buffers must be 64-byte aligned

Parameters

<i>inputL</i>	audio input left buffer (64 byte aligned)
<i>inputR</i>	audio input right buffer (64 byte aligned)
<i>output</i>	audio output mono buffer (64 byte aligned)
<i>nFrame</i>	the number of frames

8.12 Client's JSon config interfacing

Classes

- struct [bigvoice_CClientInstance](#)

Functions

- struct [bigvoice_CClientInstance](#) * [bigvoice_NewClient](#) (struct [bigvoice_CInstance](#) *instance)
- void [bigvoice_DeleteClient](#) (struct [bigvoice_CClientInstance](#) *client)
- const char * [bigvoice_ProcessJson](#) (struct [bigvoice_CClientInstance](#) *client, const char *json_str, int *need_save)
- void [bigvoice_FreeJsonAnswer](#) (const char *json_str)
- int [bigvoice_SaveState](#) (struct [bigvoice_CInstance](#) *instance)

8.12.1 Detailed Description

Allow low-level JSON configuration. see also [JSON syntax](#)

8.12.2 Function Documentation

8.12.2.1 bigvoice_DeleteClient()

```
void bigvoice_DeleteClient (
    struct bigvoice\_CClientInstance * client )
```

Deletes a client

Parameters

<i>client</i>	the client instance
---------------	---------------------

8.12.2.2 bigvoice_FreeJsonAnswer()

```
void bigvoice_FreeJsonAnswer (
    const char * json_str )
```

Free a JSON answer returned by [bigvoice_ProcessJson](#)

Parameters

<i>json_str</i>	the JSON request
-----------------	------------------

8.12.2.3 bigvoice_NewClient()

```
struct bigvoice_CClientInstance* bigvoice_NewClient (
    struct bigvoice_CInstance * instance )
```

Creates a new client

Remarks

not needed if using a port for JSON socket in [Update Management](#)

Parameters

<i>instance</i>	the processing instance
-----------------	-------------------------

Returns

A new client instance

8.12.2.4 bigvoice_ProcessJson()

```
const char* bigvoice_ProcessJson (
    struct bigvoice_CClientInstance * client,
    const char * json_str,
    int * need_save )
```

Process a JSON request

For syntax, see [JSON syntax](#)

Note

In all library, all `char*` are **UTF-8** encoded

Attention

the return value needs to be freed with [bigvoice_FreeJsonAnswer](#)

Parameters

<i>client</i>	the client instance
<i>json_str</i>	the JSON request
<i>need_save</i>	on return will be 1 if saving is required (maybe delayed)

Returns

a JSON answer (to be freed with [bigvoice_FreeJsonAnswer](#))

8.12.2.5 bigvoice_SaveState()

```
int bigvoice_SaveState (
    struct bigvoice\_CInstance * instance )
```

Save the current state

Needed when `bigvoice_ProcessJson` returned with `need_save` to 1. However it can be postponed as much as wanted.

Remarks

Works only when update thread is running

Parameters

<i>instance</i>	the processing instance
-----------------	-------------------------

Returns

0 if success, <0 on error

8.13 JSON syntax

JSON protocol to configure the process

8.13.1 Commands

```
{ "get" : 'object' }
```

get values.

If `object` is null, get all parameters, and so show the possibilities

```
{ "set" : 'object' }
```

set values.

`object` can be any subpart, only those will be changed

```
{ "change" : 'object' }
```

like get, but shows only changes since last request.

If `object` is null, get all changes.

Each call resets the state for this connection to current.

Note

In BigVoice.cl version $\leq 0.9.7$, the state resetted also for filtered out objects. So avoid alternating `object` filter between calls, or you will miss some changes.

```
{ "load" : "`presetname`" }
```

load the preset

```
{ "save" : "`presetname`" }
```

save the preset.

If null is given, save onair preset

```
{ "delete" : "`presetname`" }
```

delete the preset.

can also be an array of presetnames

```
{ "factoryload" : "`presetname`" }
```

Create a preset from factory and load it

```
{ "rename" : { "preset" : "`presetname`", "name": "`newname`" } }
```

Rename a preset

```
{ "import" : { "preset" : "`presetname`", "content": 'presetcontent' } }
```

Import a preset

```
{ "export" : "`presetname`" }
```

Export a preset: get preset's content so it can be imported later

```
{ "new" : { "name": "`presetname`", "factory": "`presetname`" } }
```

Create a new preset from a factory.

Can use array for multiple.

```
{ "undo" : 'count' }
```

Undo last(s) `count` preset change

```
{ "redo" : 'count' }
```

Redo last(s) `count` preset change

See also [JSON syntax for internal setup](#)

8.13.2 Answers

Answer is also a JSON object, starting with the command for `get` and `change`.

```
{ "get" : ... }
{ "change" : ... }
```

For other commands, if there is no error the answer is

```
{ "OK" : { "command" : "original command" } }
```

If there is an error, answer has the syntax

```
{ "ERROR" : { "command" : "faulty command", "message" : "explaining" } }
```

and may have more properties. The message may be empty.

8.13.3 Examples

8.13.3.1 Get status

```
{"get":{"status":null}}
```

answer:

```
{"get":{"status":{"preset_changed":false,"license_valid":true,"version":"1.1.19"}}
```

8.13.3.2 Get all changes since last request

```
{"change":null}
```

answer:

```
{"change":{"processorname":"Best Radio of the world",  
  "settings":{"general":{"operate":true,...},  
  "preset":{"setup":{"gate_on": true, ...},...},  
  "vu":{"setup": null,"in":{...}},  
  ...  
}}
```

8.13.3.3 Check license

```
{"get":{"status":{"license_valid":null}}}
```

answer:

```
{"get":{"status":{"license_valid":true}}}
```

8.13.3.4 Set Processor name

```
{"set":{"processorname":"Best Radio of the world"}}
```

answer:

```
{"OK":"set"}
```

8.13.3.5 Get presetlist

```
{"get":{"presetlist":null}}
```

answer:

```
{"get":{"presetlist":[{"name":"Urban","factory_name":"Urban","create":"2020-10-15 16:19:17","modify":"2020-10-15  
16:19:17"},  
  {"name":"Rock Classic","factory_name":"Rock Classic","create":"2020-10-15  
16:19:17","modify":"2020-10-15 16:19:17"},  
  {"name":"Rock","factory_name":"Rock","create":"2020-10-15 16:19:17","modify":"2020-10-15  
16:19:17"},  
  ...  
]}}
```

8.13.3.6 Load a preset

```
{"load":"Rock Classic"}
```

answer:

```
{"OK":"load"}
```

8.13.3.7 Get information on preset on air

```
{"get":{"presetinfo":null}}
```

answer:

```
{"get":{"presetinfo":{"name":"Rock Classic","factory_name":"Rock Classic","create":"2020-10-15  
16:19:17","modify":"2020-10-15 16:19:17"}}
```

8.13.3.8 Set input gain

```
{"set":{"settings":{"gain":{"in":-3.0}}}}
```

answer:

```
{"OK":"set"}
```

8.13.3.9 Set bypass

```
{"set":{"settings":{"general":{"operate":false}}}}
```

answer:

```
{"OK":"set"}
```

8.13.3.10 Export a preset

```
{"export":"MyPresetName"}
```

answer:

```
{"export":{"processor":"Best Radio of the world","presets":{"
  "MyPresetName":{"
    "presetinfo":{"name":"MyPresetName","factory_name":"Open Sound","create":"2020-10-16
06:53:20","modify":"2020-10-16 06:53:26"},
    "preset":{"setup":{"gate":-39.0}},
    "undoredo":{"undo":[{"time":"2020-10-16
06:53:23","action":{"preset":{"setup":{"gate":-30.0}}}], "redo":[], "has_undo":true, "has_redo":false}
  }}
}}
```

Note

the object `export.presets["MyPresetName"]` in the answer is the one to use as *content* in [Import a preset](#)

8.13.3.11 Import a preset

```
{"import":{"preset":"MyNewPreset","content":{"
  "presetinfo":{"name":"MyPresetName","factory_name":"Open Sound","create":"2020-10-16
06:53:20","modify":"2020-10-16 06:53:26"},
  "preset":{"setup":{"gate":-39.0}},
  "undoredo":{"undo":[{"time":"2020-10-16
06:53:23","action":{"preset":{"setup":{"gate":-30.0}}}], "redo":[], "has_undo":true, "has_redo":false}
}}
```

answer:

```
{"OK":"import"}
```

8.14 JSON syntax for internal setup

JSON commands for internal setup and information.

This is used internally for the web server.

This API might change in the future.

Those commands can be used when directly interfacing to the process (see [Client's JSON config interfacing](#)). They are blocked web going through web server (<http://.../json>)

The Authentication is effective only through the web server. The direct JSON commands are not filtered.

8.14.1 Get process and modules information

```
{ "_info": null }
```

answer:

```
{ "_info": { "product": "SOUND4 x1.CLOUD", "version": "1.3.7", "title": "x1
.CLOUD", "preset_extension": "clx1", "use_presets": true, "use_undoredo": true,
  "_modules_": [ "", "wm" ],
  "_modules_infos_": {
    "wm": { "product": "SOUND4 WM", "version": "0.1.0", "title": "SOUND4
WM", "use_presets": false, "use_undoredo": false }
  }
}
```

8.14.2 Authentication management

Some API are available only if there is a setup/admin account.

Secret stored is

```
MD5 (username+"."+realm+"."+password)
```

8.14.2.1 Difference between setup and admin account

There are 2 modes:

1. Setup mode: you define a setup account, which can then create an administrator account.
The setup account always keeps the access and can delete the administrator.
2. Admin mode: you define the administrator, which will be the single administrator.

From JSON, changing `setup_is_admin` can change the mode.

8.14.2.2 Access encoding

Access is a combinaison of letters.

- **'S'** Setup account : Allow Admin and delete admin
- **'A'** Admin access : Access to all overrides any other except setup
- **'x'** eXecute : change preset, list presets and preset infos and access settings.general fields
- **'v'** View : read preset and settings values
- **'w'** Write : change preset/settings (and undo/redo)
- **'s'** Save : create/save/delete/rename/import preset
- **'p'** Password : change own password
- **'-'** inactive : account is inactive

8.14.2.3 Change setup account

Allows to setup a setup account (even when is was not configured) or to remove it

8.14.2.3.1 Set the setup/admin account

```
{"_auth":{"init":{"setup_account":{"user":"admin","secret":"be666f22c03f199e05d3805430d69599"},"setup_is_admin":true,"u
```

answer:

```
{"OK":{"command": "_auth"}}
```

8.14.2.3.2 Set the setup/admin account with clear password

```
{"_auth":{"init":{"setup_account":{"user":"admin","secret_clear":"admin"},"setup_is_admin":true,"logged_access":"rx"}}
```

answer:

```
{"OK":{"command": "_auth"}}
```

8.14.2.3.3 Remove the setup/admin account `{"_auth":{"init":{"setup_account":null}}}`

answer:

```
{"OK":{"command": "_auth"}}
```

8.14.2.4 Get user info

```
{"_auth":{"get":{"user":"username"}}}
```

answer:

```
{"_auth":{"get":{"user":"username","secret":"secrethash","access":"rwx"}}}
```

8.14.2.5 Add or modify a user

```
{"_auth":{"add":{"user":"username","secret":"secrethash","access":"rwx"}}}
```

answer:

```
{"_auth":{"add":{"user":"username","secret":"secrethash","access":"rwx"}}}
```

8.14.2.6 Removes a user

```
{"_auth":{"remove":{"user":"username"}}}
```

answer:

```
{"_auth":{"remove":{"user":"username"}}}
```

8.14.2.7 List users

```
{"_auth":{"list":null}}
```

answer:

```
{"_auth":{"list":["user1","user2",...]}}
```

8.14.2.8 Get/set unlogged access

```
{"_auth":{"get":{"unlogged_access":null}}}
```

answer:

```
{"_auth":{"get":{"unlogged_access":"rwx"}}}
```

8.14.2.9 Get realm

```
{"_auth":{"get":{"realm":null}}}
```

answer:

```
{"_auth":{"get":{"realm":"Authentication is required"}}}
```

8.14.2.10 Get API version

```
{"_auth":{"get":{"api_version":null}}}
```

answer:

```
{"_auth":{"get":{"api_version":2}}}
```

8.14.2.11 Get setup is admin

```
{"_auth":{"get":{"setup_is_admin":null}}}
```

answer:

```
{"_auth":{"get":{"setup_is_admin":true}}}
```

8.14.2.12 Get/set setup account

```
{"_auth":{"get":{"setup_account":null}}}
```

answer:

```
{"_auth":{"get":{"setup_account":{"user":"setup","secret":"secrethash","access":"sarwx"}}}}
```

8.14.2.13 Get/set custom storage

Implementation defined extra storage, stored in auth.json

```
{"_auth":{"get":{"storage":null}}}
```

answer:

```
{"_auth":{"get":{"storage":"whateverjson..."}}
```

8.14.3 Get process' interface description

8.14.3.1 Get XML decription

```
{"_vars_xml": null}
```

answer:

```
<?xml version="1.0"?>
<Process title="x1 .CLOUD" default_name="" preset_extension="clx1">
...
</Process>
```

Attention

The answer is XML, not JSON !

8.14.3.2 Get Process Image

```
{"_image": null}
```

answer:

```
{"_image":"iVBORw0KGgoAA..." }
```

The string is the base64 encoding of the image (png format)

8.14.3.3 Get Process Ico

```
{"_ico": null}
```

answer:

```
{"_ico": "AAABAAUAEBAAAAEATABo..." }
```

The string is the base64 encoding of the icon (ico format)

8.15 Web server

Macros

- #define [SOUND4_INVALID_WEBSERVER_ID](#) ((uint64_t)-1)
- #define [SOUND4_WEBSERVER_HTTP_OK](#) (1<<0)
- #define [SOUND4_WEBSERVER_HTTPS_OK](#) (1<<1)

Functions

- uint64_t [bigvoice_Webserver_tcp](#) (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port)
- uint64_t [bigvoice_Webserver_tcp2](#) (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port, const struct [bigvoice_CParameters](#) *parameters)
- uint64_t [bigvoice_Webserver](#) (unsigned int listenport, unsigned int listenport_secure, struct [bigvoice_CInstance](#) *instance)
- int [bigvoice_Webserver_Stop](#) (uint64_t id, int timeout_ms)
- int [bigvoice_Webserver_Status](#) (uint64_t id)

8.15.1 Detailed Description

Run a web server to configure the processor

See also how to setup authentication with [Environment variables and parameters](#). Their use is documented in [bigvoice_InitProcess](#).

8.15.2 Macro Definition Documentation

8.15.2.1 SOUND4_INVALID_WEBSERVER_ID

```
#define SOUND4_INVALID_WEBSERVER_ID ((uint64_t)-1)
```

This represents an invalid webserver id [bigvoice_Webserver](#) and [bigvoice_Webserver_tcp](#) will never return this value

Definition at line [1502](#) of file [sound4.bigvoice.cl.h](#).

8.15.2.2 SOUND4_WEBSERVER_HTTP_OK

```
#define SOUND4_WEBSERVER_HTTP_OK (1<<0)
```

If this bit is set in the return value from [bigvoice_Webserver_Status](#), then HTTP server is OK

Definition at line 1507 of file [sound4.bigvoice.cl.h](#).

8.15.2.3 SOUND4_WEBSERVER_HTTPS_OK

```
#define SOUND4_WEBSERVER_HTTPS_OK (1<<1)
```

If this bit is set in the return value from [bigvoice_Webserver_Status](#), then HTTPS server is OK

Definition at line 1511 of file [sound4.bigvoice.cl.h](#).

8.15.3 Function Documentation

8.15.3.1 bigvoice_Webserver()

```
uint64_t bigvoice_Webserver (
    unsigned int listenport,
    unsigned int listenport_secure,
    struct bigvoice\_CInstance * instance )
```

Run a web server, connecting to local instance internally (no expose port needed)

This function returns after connecting to the process, and continue to run in background See [bigvoice_Webserver_Stop](#) too stop it gracefully

For secure connection, it will use `privatekey.pem` and `certificate.pem` from current directory. Environment variables `SSL_PRIVATE_PATH` and `SSL_CERTIFICATE_PATH` can override those (see [Environment variables and parameters](#)). If none is valid, it will try also `/etc/ssl/certs/server.key` and `/etc/ssl/certs/server.crt`.

Warning

On Windows, calling this from Go<1.20 code will crash.

Parameters

<i>listenport</i>	The port to listen for HTTP connections (0 to disable)
<i>listenport_secure</i>	The port to listen for HTTPS connections (0 to disable)
<i>instance</i>	The process instance to connect to

Returns

id of webserver (starting at 0), to use with `bigvoice_Webserver_Stop`

8.15.3.2 bigvoice_Webserver_Status()

```
int bigvoice_Webserver_Status (
    uint64_t id )
```

Get the web server status

Warning

On Windows, calling this from Go<1.20 code will crash.

Parameters

<i>id</i>	The web server Id, returned by bigvoice_Webserver_tcp or bigvoice_Webserver
-----------	---

Returns

a combinaison of [SOUND4_WEBSERVER_HTTP_OK](#) and [SOUND4_WEBSERVER_HTTPS_OK](#) depending on the server status

8.15.3.3 bigvoice_Webserver_Stop()

```
int bigvoice_Webserver_Stop (
    uint64_t id,
    int timeout_ms )
```

Stop a web server started with [bigvoice_Webserver_tcp](#)

Warning

On Windows, calling this from Go<1.20 code will crash.

Parameters

<i>id</i>	The web server Id, returned by bigvoice_Webserver_tcp or bigvoice_Webserver
<i>timeout_ms</i>	The timeout to wait gracefull shutdown

Returns

0 if succeed, -1 if the id is not found

8.15.3.4 bigvoice_Webserver_tcp()

```
uint64_t bigvoice_Webserver_tcp (
    unsigned int listenport,
    unsigned int listenport_secure,
    const char * socket_ip,
    unsigned int socket_port )
```

Run a web server, connecting to process with JSON over TCP

This function returns after connecting to the process, and continue to run in background See [bigvoice_Webserver_Stop](#) too stop it gracefully

For secure connection, it will use `privatekey.pem` and `certificate.pem` from current directory. Environment variables `SSL_PRIVATE_PATH` and `SSL_CERTIFICATE_PATH` can override those (see [Environment variables and parameters](#)). If none is valid, it will try also `/etc/ssl/certs/server.key` and `/etc/ssl/certs/server.crt`.

Warning

On Windows, calling this from Go<1.20 code will crash.

Parameters

<i>listenport</i>	The port to listen for HTTP connections
<i>listenport_secure</i>	The port to listen for HTTPS connections
<i>socket_ip</i>	The ip of the process to connect to
<i>socket_port</i>	The port of the process's JSON server to connect to

Returns

id of webserver (starting at 0), to use with `bigvoice_Webserver_Stop`

8.15.3.5 bigvoice_Webserver_tcp2()

```
uint64_t bigvoice_Webserver_tcp2 (
    unsigned int listenport,
    unsigned int listenport_secure,
    const char * socket_ip,
    unsigned int socket_port,
    const struct bigvoice\_CParameters * parameters )
```

Run a web server, connecting to process with JSON over TCP, using parameters

This function returns after connecting to the process, and continue to run in background See [bigvoice_Webserver_Stop](#) too stop it gracefully

For secure connection, it will use `privatekey.pem` and `certificate.pem` from current directory. Environment variables `SSL_PRIVATE_PATH` and `SSL_CERTIFICATE_PATH` can override those (see [Environment variables and parameters](#)). If none is valid, it will try also `/etc/ssl/certs/server.key` and `/etc/ssl/certs/server.crt`.

Warning

On Windows, calling this from Go<1.20 code will crash.

Parameters

<i>listenport</i>	The port to listen for HTTP connections
<i>listenport_secure</i>	The port to listen for HTTPS connections
<i>socket_ip</i>	The ip of the process to connect to
<i>socket_port</i>	The port of the process's JSON server to connect to
<i>parameters</i>	The parameters to use, created with bigvoice_NewParameters

Returns

id of webserver (starting at 0), to use with `bigvoice_Webserver_Stop`

8.16 Sample using C++ interface wrapper

See full example [cpp/sound4.bigvoice.cl-pipe-cxx.cpp](#)

8.16.1 Setup and sanity checks

```
sound4::bigvoice::SetLogSeverity(verbose5);
sound4::bigvoice::SetLoggerCallback(MyLogger);
sound4::bigvoice::CInstance instance;
instance.SetParam("ADMIN_USER", "admin");
instance.SetParam("ADMIN_SECRET", "admin");
```

8.16.2 Sample to create the processing instance

```
if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, store_path)) {
    std::cerr << "Failed to create the processing instance" << std::endl;
    return 1;
}
```

8.16.3 Communicating with the instance

See also

JSON syntax

```
auto client = instance.NewClient();
auto answer = client->ProcessJson(R"JSON({"get":{"processorname":null}})JSON");
std::cerr << "Request for processor name returned " << answer << std::endl;
```

8.16.4 Starting a web-server

```
if (!instance.StartWebServer(listen_port, 0)) {
    std::cerr << "Failed to create the web server on port " << listen_port << std::endl;
    return 1;
}
```

8.16.5 Sample processing loop

```
while (!stopthread)
{
    // Read a chunk from stdin
    if (std::fread(instance.GetBufferIn().data(), sizeof(float), sound4::bigvoice::InputSampleSize,
        stdin) != sound4::bigvoice::InputSampleSize) {
        std::cerr << "ProcessLoop: input stopped" << std::endl;
        stopthread=true;
        break;
    }
    if (stopthread) break;
    // Process it
    instance.ProcessAudio();
    // Write it to stdout
    if (fwrite(instance.GetBufferOut().data(), sizeof(float), sound4::bigvoice::OutputSampleSize,
        stdout) != sound4::bigvoice::OutputSampleSize) {
        std::cerr << "Failed to write audio output" << std::endl;
        stopthread=true;
    }
}
```

8.17 Sample using C++ interface wrapper with dynamic library loading

See full example [cpp/sound4.bigvoice.cl-pipe-dyn.cpp](#)

8.17.1 Dynamic loading

```
sound4::bigvoice::dyn::CDynLib dynlib;
if (!dynlib.Load()) {
    std::cerr << "Failed to load processing library" << std::endl;
    return 1;
}
```

8.17.2 Setup and sanity checks

```
sound4::bigvoice::dyn::SetLogSeverity(dynlib, verbose5);
sound4::bigvoice::dyn::SetLoggerCallback(dynlib, MyLogger);
sound4::bigvoice::dyn::CInstance instance(dynlib);
instance.SetParam("ADMIN_USER", "admin");
instance.SetParam("ADMIN_SECRET", "admin");
```

8.17.3 Sample to create the processing instance

```
if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, {})) {
    std::cerr << "Failed to create the processing instance" << std::endl;
    return 1;
}
```

8.17.4 Communicating with the instance

See also

JSON syntax

```
auto client = instance.NewClient();
auto answer = client->ProcessJson(R"JSON({"get":{"processorname":null}})JSON");
std::cerr << "Request for processor name returned " << answer << std::endl;
```

8.17.5 Starting a web-server

```
if (!instance.StartWebServer(listen_port, 0)) {
    std::cerr << "Failed to create the web server on port " << listen_port << std::endl;
    return 1;
}
```

8.17.6 Sample processing loop

```

while (!stopthread)
{
    // Read a chunk from stdin
    if (std::fread(instance.GetBufferIn().data(), sizeof(float), sound4::bigvoice::dyn::InputSampleSize,
        stdin) != sound4::bigvoice::dyn::InputSampleSize) {
        std::cerr << "ProcessLoop: input stopped" << std::endl;
        stopthread=true;
        break;
    }
    if (stopthread) break;
    // Process it
    instance.ProcessAudio();
    // Write it to stdout
    if (fwrite(instance.GetBufferOut().data(), sizeof(float), sound4::bigvoice::dyn::OutputSampleSize,
        stdout) != sound4::bigvoice::dyn::OutputSampleSize) {
        std::cerr << "Failed to write audio output" << std::endl;
        stopthread=true;
    }
}

```

8.17.7 Custom preset manager

```

class MyPresetManager : public sound4::bigvoice::dyn::CPresetLoader {
public:
    MyPresetManager() = default;
    virtual ~MyPresetManager() = default;
    bool SetupPath(const std::filesystem::path& a_preset_path)
    {
        m_preset_path=a_preset_path;
        if ( !std::filesystem::exists(m_preset_path) ) {
            return false;
        }
        std::error_code ec;
        if ( !std::filesystem::is_directory(m_preset_path, ec) ) {
            return false;
        }
        return true;
    }
    virtual bool IsReadOnly() override { return false; };
    virtual bool Exists(const std::filesystem::path &name) override
    {
        std::error_code ec;
        return std::filesystem::exists(GetRealPath(name), ec);
    }
    virtual bool Remove(const std::filesystem::path &name) override
    {
        std::error_code ec;
        return std::filesystem::remove(GetRealPath(name), ec);
    }
    virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) override
    {
        std::error_code ec;
        std::filesystem::rename(GetRealPath(from), GetRealPath(to), ec);
        return (!ec);
    }
    virtual std::vector<std::filesystem::path> GetAll() override
    {
        std::vector<std::filesystem::path> list;
        for (const auto & entry : std::filesystem::directory_iterator(m_preset_path.c_str(),
            std::filesystem::directory_options::skip_permission_denied)) {
            if (entry.is_regular_file()) {
                list.push_back(std::filesystem::relative(entry.path(), m_preset_path).native());
            }
        }
        return list;
    }
    virtual std::string Read(const std::filesystem::path &filename) override
    {
        std::ifstream f(GetRealPath(filename));
        if ( f.fail() ) {
            return std::string();
        }
        std::ostringstream ss;
        ss << f.rdbuf();
        return ss.str();
    }
    virtual bool Write(const std::filesystem::path &filename, const std::string &content) override
    {
        std::ofstream f(GetRealPath(filename));
        if ( f.fail() ) {
            return false;
        }
    }
}

```

```
        f « content;
        if ( f.fail() ) {
            return false;
        }
        return true;
    }
private:
    std::filesystem::path m_preset_path;
    std::filesystem::path GetRealPath(const std::filesystem::path &name)
    {
        return m_preset_path / name;
    }
};
```

8.17.8 Custom preset manager loading

```
MyPresetManager preset_manager;
if (!store_path.empty()) {
    if (!preset_manager.SetupPath(store_path)) {
        std::cerr « "Failed to setup the custom preset manager on path " « store_path.c_str() «
        std::endl;
        return 1;
    }
    instance.SetPresetManager(&preset_manager);
}
```


Chapter 9

Namespace Documentation

9.1 dyn Namespace Reference

9.1.1 Detailed Description

Dynamic loading namespace

9.2 sound4 Namespace Reference

Namespaces

- [bigvoice](#)

9.2.1 Detailed Description

SOUND4 namespace

9.3 sound4::bigvoice Namespace Reference

Namespaces

- [dyn](#)
- [helper](#)

Classes

- class [CBus](#)
- class [CPresetLoader](#)
- class [CInstance](#)

Typedefs

- using [SampleFormat](#) = [bigvoice_SampleFormat](#)
- using [LogSeverity](#) = [bigvoice_LogSeverity](#)
- using [log_cb_t](#) = std::function< void([LogSeverity](#), const std::string &)>
- using [PresetChange_Kind](#) = [bigvoice_PresetChange_Kind](#)

Variables

- const size_t [ChunkFrames](#) = [BIGVOICE_AUDIOFRAME_COUNT](#)
- const size_t [InputChannels](#) = [BIGVOICE_CHANNEL_COUNT](#)*[BIGVOICE_AUDIO_INPUT_COUNT](#)
- const size_t [OutputChannels](#) = [BIGVOICE_CHANNEL_COUNT](#)*[BIGVOICE_AUDIO_OUTPUT_COUNT](#)
- const size_t [InputSampleSize](#) = [ChunkFrames](#)*[InputChannels](#)
- const size_t [OutputSampleSize](#) = [ChunkFrames](#)*[OutputChannels](#)
- const size_t [SampleRate](#) = [BIGVOICE_SAMPLE_RATE](#)

9.3.1 Detailed Description

bigvoice namespace

9.3.2 Typedef Documentation

9.3.2.1 log_cb_t

```
using sound4::bigvoice::log\_cb\_t = typedef std::function<void(LogSeverity,const std::string&)>
```

Log callback type

Definition at line [320](#) of file [sound4.bigvoice.cl.hpp](#).

9.3.2.2 LogSeverity

```
using sound4::bigvoice::LogSeverity = typedef bigvoice\_LogSeverity
```

Log Severity levels

Definition at line [316](#) of file [sound4.bigvoice.cl.hpp](#).

9.3.2.3 PresetChange_Kind

```
using sound4::bigvoice::PresetChange_Kind = typedef bigvoice_PresetChange_Kind
```

Definition at line 380 of file [sound4.bigvoice.cl.hpp](#).

9.3.2.4 SampleFormat

```
using sound4::bigvoice::SampleFormat = typedef bigvoice_SampleFormat
```

Definition at line 53 of file [sound4.bigvoice.cl.hpp](#).

9.3.3 Variable Documentation

9.3.3.1 ChunkFrames

```
const size_t sound4::bigvoice::ChunkFrames =BIGVOICE_AUDIOFRAME_COUNT
```

Number of frames in a chunk

Definition at line 57 of file [sound4.bigvoice.cl.hpp](#).

9.3.3.2 InputChannels

```
const size_t sound4::bigvoice::InputChannels =BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_INPUT_COUNT
```

Number of float values in an input frame

Definition at line 61 of file [sound4.bigvoice.cl.hpp](#).

9.3.3.3 InputSampleSize

```
const size_t sound4::bigvoice::InputSampleSize =ChunkFrames*InputChannels
```

Number of float values for an input chunk

Definition at line 69 of file [sound4.bigvoice.cl.hpp](#).

9.3.3.4 OutputChannels

```
const size_t sound4::bigvoice::OutputChannels =BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_OUTPUT_COUNT
```

Number of float values in an output frame

Definition at line 65 of file [sound4.bigvoice.cl.hpp](#).

9.3.3.5 OutputSampleSize

```
const size_t sound4::bigvoice::OutputSampleSize =ChunkFrames*OutputChannels
```

Number of float values for an output chunk

Definition at line 73 of file [sound4.bigvoice.cl.hpp](#).

9.3.3.6 SampleRate

```
const size_t sound4::bigvoice::SampleRate =BIGVOICE_SAMPLE_RATE
```

Sample rate of library

Definition at line 77 of file [sound4.bigvoice.cl.hpp](#).

9.4 sound4::bigvoice::dyn Namespace Reference

Namespaces

- [helper](#)

Classes

- class [CDynLib](#)
Dynamic library interface.
- class [CBus](#)
- class [CPresetLoader](#)
- class [CInstance](#)

Typedefs

- using [SampleFormat](#) = [bigvoice_SampleFormat](#)
- using [LogSeverity](#) = [bigvoice_LogSeverity](#)
- using [log_cb_t](#) = std::function< void([LogSeverity](#), const std::string &)>
- using [PresetChange_Kind](#) = [bigvoice_PresetChange_Kind](#)

Variables

- `const size_t ChunkFrames = BIGVOICE_AUDIOFRAME_COUNT`
- `const size_t InputChannels = BIGVOICE_CHANNEL_COUNT * BIGVOICE_AUDIO_INPUT_COUNT`
- `const size_t OutputChannels = BIGVOICE_CHANNEL_COUNT * BIGVOICE_AUDIO_OUTPUT_COUNT`
- `const size_t InputSampleSize = ChunkFrames * InputChannels`
- `const size_t OutputSampleSize = ChunkFrames * OutputChannels`
- `const size_t SampleRate = BIGVOICE_SAMPLE_RATE`

9.4.1 Typedef Documentation

9.4.1.1 log_cb_t

```
using sound4::bigvoice::dyn::log_cb_t = typedef std::function<void(LogSeverity, const std::string&)>
```

Log callback type

Definition at line 697 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.1.2 LogSeverity

```
using sound4::bigvoice::dyn::LogSeverity = typedef bigvoice_LogSeverity
```

Log Severity levels

Definition at line 693 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.1.3 PresetChange_Kind

```
using sound4::bigvoice::dyn::PresetChange_Kind = typedef bigvoice_PresetChange_Kind
```

Definition at line 765 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.1.4 SampleFormat

```
using sound4::bigvoice::dyn::SampleFormat = typedef bigvoice_SampleFormat
```

Definition at line 426 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.2 Variable Documentation

9.4.2.1 ChunkFrames

```
const size_t sound4::bigvoice::dyn::ChunkFrames =BIGVOICE_AUDIOFRAME_COUNT
```

Number of frames in a chunk

Definition at line 430 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.2.2 InputChannels

```
const size_t sound4::bigvoice::dyn::InputChannels =BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_INPUT_COUNT
```

Number of float values in an input frame

Definition at line 434 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.2.3 InputSampleSize

```
const size_t sound4::bigvoice::dyn::InputSampleSize =ChunkFrames*InputChannels
```

Number of float values for an input chunk

Definition at line 442 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.2.4 OutputChannels

```
const size_t sound4::bigvoice::dyn::OutputChannels =BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_OUTPUT_COUNT
```

Number of float values in an output frame

Definition at line 438 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.2.5 OutputSampleSize

```
const size_t sound4::bigvoice::dyn::OutputSampleSize =ChunkFrames*OutputChannels
```

Number of float values for an output chunk

Definition at line 446 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.4.2.6 SampleRate

```
const size_t sound4::bigvoice::dyn::SampleRate =BIGVOICE_SAMPLE_RATE
```

Sample rate of library

Definition at line 450 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.5 sound4::bigvoice::dyn::helper Namespace Reference

Classes

- class [DynFuncHolder](#)
Helper to wrap a function pointer.
- class [CDynLoader](#)
Helper to load dynamic library.
- struct [SampleFormat](#)
Helper for Sample format types.
- struct [SampleFormat< int16_t >](#)
- struct [SampleFormat< int32_t >](#)
- struct [SampleFormat< float >](#)

Typedefs

- using [DynLib_t](#) = HMODULE
- using [DynFunc_t](#) = FARPROC

9.5.1 Detailed Description

helper functions for unicode strings in Windows

9.5.2 Typedef Documentation

9.5.2.1 DynFunc_t

```
using sound4::bigvoice::dyn::helper::DynFunc_t = typedef FARPROC
```

Definition at line 77 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.5.2.2 DynLib_t

```
using sound4::bigvoice::dyn::helper::DynLib_t = typedef HMODULE
```

Definition at line 76 of file [sound4.bigvoice.cl_dyn.hpp](#).

9.6 sound4::bigvoice::helper Namespace Reference

Classes

- struct [SampleFormat](#)
Helper for Sample format types.
- struct [SampleFormat< int16_t >](#)
- struct [SampleFormat< int32_t >](#)
- struct [SampleFormat< float >](#)

9.6.1 Detailed Description

helper functions for unicode strings in Windows

9.7 std Namespace Reference

Chapter 10

Class Documentation

10.1 bigvoice_CBus Struct Reference

```
#include <sound4.bigvoice.cl.h>
```

10.1.1 Detailed Description

Abstract holder for a instances' bus

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl.h](#)

10.2 bigvoice_CClientInstance Struct Reference

```
#include <sound4.bigvoice.cl.h>
```

10.2.1 Detailed Description

Abstract pointer for a client instance

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl.h](#)

10.3 bigvoice_CInstance Struct Reference

```
#include <sound4.bigvoice.cl.h>
```

10.3.1 Detailed Description

Abstract holder for a process instance

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl.h](#)

10.4 bigvoice_CParameters Struct Reference

```
#include <sound4.bigvoice.cl.h>
```

10.4.1 Detailed Description

Abstract holder for a process instance parameters

Definition at line 307 of file [sound4.bigvoice.cl.h](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl.h](#)

10.5 sound4::bigvoice::CBus Class Reference

```
#include <sound4.bigvoice.cl.hpp>
```

Public Member Functions

- [CBus](#) ()
- [~CBus](#) ()
- [bigvoice_CBus](#) * [Get](#) () const

10.5.1 Detailed Description

Definition at line 348 of file [sound4.bigvoice.cl.hpp](#).

10.5.2 Constructor & Destructor Documentation

10.5.2.1 CBus()

```
sound4::bigvoice::CBus::CBus ( ) [inline]
```

Definition at line 350 of file [sound4.bigvoice.cl.hpp](#).

10.5.2.2 ~CBus()

```
sound4::bigvoice::CBus::~CBus ( ) [inline]
```

Definition at line 352 of file [sound4.bigvoice.cl.hpp](#).

10.5.3 Member Function Documentation

10.5.3.1 Get()

```
bigvoice_CBus* sound4::bigvoice::CBus::Get ( ) const [inline]
```

Definition at line 357 of file [sound4.bigvoice.cl.hpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl.hpp](#)

10.6 sound4::bigvoice::dyn::CBus Class Reference

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Public Member Functions

- [CBus](#) (CDynLib &dynlib)
- [~CBus](#) ()
- [bigvoice_CBus](#) * [Get](#) () const

10.6.1 Detailed Description

Definition at line 725 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.6.2 Constructor & Destructor Documentation

10.6.2.1 CBus()

```
sound4::bigvoice::dyn::CBus::CBus (
    CDynLib & dynlib ) [inline]
```

Definition at line 727 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.6.2.2 ~CBus()

```
sound4::bigvoice::dyn::CBus::~~CBus ( ) [inline]
```

Definition at line 734 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.6.3 Member Function Documentation

10.6.3.1 Get()

```
bigvoice_CBus* sound4::bigvoice::dyn::CBus::Get ( ) const [inline]
```

Definition at line 741 of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.7 sound4::bigvoice::CInstance::CClient Class Reference

```
#include <sound4.bigvoice.cl.hpp>
```

Public Member Functions

- [CClient](#) ([bigvoice_CInstance](#) *instance)
- [CClient](#) (const [CInstance](#) &)=delete
- [CClient](#) & [operator=](#) (const [CInstance](#) &)=delete
- [~CClient](#) ()
- std::string [ProcessJson](#) (const std::string &request, bool *NeedSave=nullptr)
Process a JSON request and returns the answer.

10.7.1 Detailed Description

Low-level JSON client.

See also

[JSON syntax](#)

Definition at line 770 of file [sound4.bigvoice.cl.hpp](#).

10.7.2 Constructor & Destructor Documentation

10.7.2.1 CClient() [1/2]

```
sound4::bigvoice::CInstance::CClient::CClient (
    bigvoice_CInstance * instance ) [inline]
```

Definition at line 772 of file [sound4.bigvoice.cl.hpp](#).

10.7.2.2 CClient() [2/2]

```
sound4::bigvoice::CInstance::CClient::CClient (
    const CInstance & ) [delete]
```

10.7.2.3 ~CClient()

```
sound4::bigvoice::CInstance::CClient::~~CClient ( ) [inline]
```

Definition at line 781 of file [sound4.bigvoice.cl.hpp](#).

10.7.3 Member Function Documentation

10.7.3.1 operator=()

```
CClient& sound4::bigvoice::CInstance::CClient::operator= (
    const CInstance & ) [delete]
```

10.7.3.2 ProcessJson()

```
std::string sound4::bigvoice::CInstance::CClient::ProcessJson (
    const std::string & request,
    bool * NeedSave = nullptr ) [inline]
```

Parameters

<i>request</i>	the JSON request.
----------------	-------------------

See also

[JSON syntax](#)

Parameters

<i>NeedSave</i>	if non null, returns if the saving is required (maybe delayed)
-----------------	--

Returns

the JSON answer

Definition at line 795 of file [sound4.bigvoice.cl.hpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl.hpp](#)

10.8 sound4::bigvoice::dyn::CInstance::CClient Class Reference

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Public Member Functions

- [CClient](#) ([bigvoice_CInstance](#) *instance, [CDynLib](#) &dynlib)
- [CClient](#) (const [CInstance](#) &)=delete
- [CClient](#) & [operator=](#) (const [CInstance](#) &)=delete
- [~CClient](#) ()
- std::string [ProcessJson](#) (const std::string &request, bool *NeedSave=nullptr)
Process a JSON request and returns the answer.

10.8.1 Detailed Description

Low-level JSON client.

See also

[JSON syntax](#)

Definition at line 1170 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.8.2 Constructor & Destructor Documentation

10.8.2.1 CClient() [1/2]

```
sound4::bigvoice::dyn::CInstance::CClient::CClient (
    bigvoice_CInstance * instance,
    CDynLib & dynlib ) [inline]
```

Definition at line 1172 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.8.2.2 CClient() [2/2]

```
sound4::bigvoice::dyn::CInstance::CClient::CClient (
    const CInstance & ) [delete]
```

10.8.2.3 ~CClient()

```
sound4::bigvoice::dyn::CInstance::CClient::~~CClient ( ) [inline]
```

Definition at line 1182 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.8.3 Member Function Documentation

10.8.3.1 operator=()

```
CClient& sound4::bigvoice::dyn::CInstance::CClient::operator= (
    const CInstance & ) [delete]
```

10.8.3.2 ProcessJson()

```
std::string sound4::bigvoice::dyn::CInstance::CClient::ProcessJson (
    const std::string & request,
    bool * NeedSave = nullptr ) [inline]
```

Parameters

<i>request</i>	the JSON request.
----------------	-------------------

See also

[JSON syntax](#)

Parameters

<i>NeedSave</i>	if non null, returns if the saving is required (maybe delayed)
-----------------	--

Returns

the JSON answer

Definition at line 1196 of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.9 sound4::bigvoice::dyn::CDynLib Class Reference

Dynamic library interface.

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Public Member Functions

- [CDynLib](#) ()=default
- bool [Load](#) (const std::filesystem::path &path=std::filesystem::current_path())
Loads the library.
- bool [IsOk](#) () const
Check if the library was loaded correctly.

Public Attributes

- [helper::DynFuncHolder](#)< decltype(bigvoice_GetVersion) > [GetVersion](#)
- [helper::DynFuncHolder](#)< decltype(bigvoice_GetChunkSizeInFrames) > [GetChunkSizeInFrames](#)
- [helper::DynFuncHolder](#)< decltype(bigvoice_GetChannelCount) > [GetChannelCount](#)
- [helper::DynFuncHolder](#)< decltype(bigvoice_GetAudioInputCount) > [GetAudioInputCount](#)
- [helper::DynFuncHolder](#)< decltype(bigvoice_GetAudioOutputCount) > [GetAudioOutputCount](#)
- [helper::DynFuncHolder](#)< decltype(bigvoice_GetSampleRate) > [GetSampleRate](#)
- [helper::DynFuncHolder](#)< decltype(bigvoice_SetLoggerCallback) > [SetLoggerCallback](#)
- [helper::DynFuncHolder](#)< decltype(bigvoice_SetLogSeverity) > [SetLogSeverity](#)
- [helper::DynFuncHolder](#)< decltype(bigvoice_NewParameters) > [NewParameters](#)

- [helper::DynFuncHolder< decltype\(bigvoice_FreeParameters\) > FreeParameters](#)
- [helper::DynFuncHolder< decltype\(bigvoice_SetParameter\) > SetParameter](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetParameter\) > GetParameter](#)
- [helper::DynFuncHolder< decltype\(bigvoice_FreeParameterValue\) > FreeParameterValue](#)
- [helper::DynFuncHolder< decltype\(bigvoice_InitProcess\) > InitProcess](#)
- [helper::DynFuncHolder< decltype\(bigvoice_InitProcess2\) > InitProcess2](#)
- [helper::DynFuncHolder< decltype\(bigvoice_TerminateProcess\) > TerminateProcess](#)
- [helper::DynFuncHolder< decltype\(bigvoice_ExitProcess\) > ExitProcess](#)
- [helper::DynFuncHolder< decltype\(bigvoice_StartUpdateThread\) > StartUpdateThread](#)
- [helper::DynFuncHolder< decltype\(bigvoice_StopUpdateThread\) > StopUpdateThread](#)
- [helper::DynFuncHolder< decltype\(bigvoice_WaitUpdateThreadReady\) > WaitUpdateThreadReady](#)
- [helper::DynFuncHolder< decltype\(bigvoice_ProcessAudio\) > ProcessAudio](#)
- [helper::DynFuncHolder< decltype\(bigvoice_ProcessAudio_Planar\) > ProcessAudio_Planar](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetBufferIn\) > GetBufferIn](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetBufferOut\) > GetBufferOut](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetEstimatedDelay\) > GetEstimatedDelay](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetFormatName\) > GetFormatName](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetFormatFromName\) > GetFormatFromName](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetBytesFromFormat\) > GetBytesFromFormat](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetMaxPacketFrame\) > GetMaxPacketFrame](#)
- [helper::DynFuncHolder< decltype\(bigvoice_AddAudio\) > AddAudio](#)
- [helper::DynFuncHolder< decltype\(bigvoice_AddPadAudio\) > AddPadAudio](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetOutputCount\) > GetOutputCount](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetAudio\) > GetAudio](#)
- [helper::DynFuncHolder< decltype\(bigvoice_AudioConvertFrom\) > AudioConvertFrom](#)
- [helper::DynFuncHolder< decltype\(bigvoice_AudioConvertTo\) > AudioConvertTo](#)
- [helper::DynFuncHolder< decltype\(bigvoice_StereoToMono\) > StereoToMono](#)
- [helper::DynFuncHolder< decltype\(bigvoice_MonoToStereo\) > MonoToStereo](#)
- [helper::DynFuncHolder< decltype\(bigvoice_AudioMonoFromLiveStereo\) > AudioMonoFromLiveStereo](#)
- [helper::DynFuncHolder< decltype\(bigvoice_AudioMonoToLiveStereo\) > AudioMonoToLiveStereo](#)
- [helper::DynFuncHolder< decltype\(bigvoice_NewClient\) > NewClient](#)
- [helper::DynFuncHolder< decltype\(bigvoice_DeleteClient\) > DeleteClient](#)
- [helper::DynFuncHolder< decltype\(bigvoice_ProcessJson\) > ProcessJson](#)
- [helper::DynFuncHolder< decltype\(bigvoice_FreeJsonAnswer\) > FreeJsonAnswer](#)
- [helper::DynFuncHolder< decltype\(bigvoice_SaveState\) > SaveState](#)
- [helper::DynFuncHolder< decltype\(bigvoice_Webserver_tcp\) > Webserver_tcp](#)
- [helper::DynFuncHolder< decltype\(bigvoice_Webserver_tcp2\) > Webserver_tcp2](#)
- [helper::DynFuncHolder< decltype\(bigvoice_Webserver\) > Webserver](#)
- [helper::DynFuncHolder< decltype\(bigvoice_Webserver_Stop\) > Webserver_Stop](#)
- [helper::DynFuncHolder< decltype\(bigvoice_Webserver_Status\) > Webserver_Status](#)
- [helper::DynFuncHolder< decltype\(bigvoice_StereoToMono_Planar\) > StereoToMono_Planar](#)
- [helper::DynFuncHolder< decltype\(bigvoice_MonoToStereo_Planar\) > MonoToStereo_Planar](#)
- [helper::DynFuncHolder< decltype\(bigvoice_SetMetadata\) > SetMetadata](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetMetadataInfos\) > GetMetadataInfos](#)
- [helper::DynFuncHolder< decltype\(bigvoice_FreeMetadataInfos\) > FreeMetadataInfos](#)
- [helper::DynFuncHolder< decltype\(bigvoice_SetPresetManager\) > SetPresetManager](#)
- [helper::DynFuncHolder< decltype\(bigvoice_PresetManager_InformChange\) > PresetManager_InformChange](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetPossibleChunkSizeInFrames\) > GetPossibleChunkSizeInFrames](#)
- [helper::DynFuncHolder< decltype\(bigvoice_GetProcessChunkFrames\) > GetProcessChunkFrames](#)
- [helper::DynFuncHolder< decltype\(bigvoice_InitProcess3\) > InitProcess3](#)
- [helper::DynFuncHolder< decltype\(bigvoice_NewBus\) > NewBus](#)
- [helper::DynFuncHolder< decltype\(bigvoice_FreeBus\) > FreeBus](#)
- [helper::DynFuncHolder< decltype\(bigvoice_SetInstanceBus\) > SetInstanceBus](#)

10.9.1 Detailed Description

Once you loaded the library with [Load\(\)](#), if it succeed you can use all the standard C functions from this library.

Definition at line 210 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.2 Constructor & Destructor Documentation

10.9.2.1 CDynLib()

```
sound4::bigvoice::dyn::CDynLib::CDynLib ( ) [default]
```

10.9.3 Member Function Documentation

10.9.3.1 IsOk()

```
bool sound4::bigvoice::dyn::CDynLib::IsOk ( ) const [inline]
```

Returns

true if OK.
false if not OK.

Definition at line 416 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.3.2 Load()

```
bool sound4::bigvoice::dyn::CDynLib::Load (
    const std::filesystem::path & path = std::filesystem::current_path() ) [inline]
```

Parameters

<i>path</i>	the path of the library to load.
-------------	----------------------------------

Returns

true if library was loaded correctly.
false if library load failed. Do not use in this case !

Definition at line 302 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4 Member Data Documentation

10.9.4.1 AddAudio

```
helper::DynFuncHolder< decltype(bigvoice_AddAudio) > sound4::bigvoice::dyn::CDynLib::AddAudio
```

Definition at line 247 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.2 AddPadAudio

```
helper::DynFuncHolder< decltype(bigvoice_AddPadAudio) > sound4::bigvoice::dyn::CDynLib::AddPadAudio
```

Definition at line 248 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.3 AudioConvertFrom

```
helper::DynFuncHolder< decltype(bigvoice_AudioConvertFrom) > sound4::bigvoice::dyn::CDynLib::AudioConvertFrom
```

Definition at line 251 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.4 AudioConvertTo

```
helper::DynFuncHolder< decltype(bigvoice_AudioConvertTo) > sound4::bigvoice::dyn::CDynLib::AudioConvertTo
```

Definition at line 252 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.5 AudioMonoFromLiveStereo

```
helper::DynFuncHolder< decltype(bigvoice_AudioMonoFromLiveStereo) > sound4::bigvoice::dyn::CDynLib::AudioMonoFromLiveStereo
```

Definition at line 255 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.6 AudioMonoToLiveStereo

```
helper::DynFuncHolder< decltype(bigvoice_AudioMonoToLiveStereo) > sound4::bigvoice::dyn::CDynLib::AudioMonoToLiveStereo
```

Definition at line 256 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.7 DeleteClient

```
helper::DynFuncHolder< decltype(bigvoice_DeleteClient) > sound4::bigvoice::dyn::CDynLib::DeleteClient
```

Definition at line 258 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.8 ExitProcess

```
helper::DynFuncHolder< decltype(bigvoice_ExitProcess) > sound4::bigvoice::dyn::CDynLib::ExitProcess
```

Definition at line 234 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.9 FreeBus

```
helper::DynFuncHolder< decltype(bigvoice_FreeBus) > sound4::bigvoice::dyn::CDynLib::FreeBus
```

Definition at line 291 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.10 FreeJsonAnswer

```
helper::DynFuncHolder< decltype(bigvoice_FreeJsonAnswer) > sound4::bigvoice::dyn::CDynLib::FreeJsonAnswer
```

Definition at line 260 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.11 FreeMetadataInfos

```
helper::DynFuncHolder< decltype(bigvoice_FreeMetadataInfos) > sound4::bigvoice::dyn::CDynLib::FreeMetadataInfos
```

Definition at line 277 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.12 FreeParameters

```
helper::DynFuncHolder< decltype(bigvoice_FreeParameters) > sound4::bigvoice::dyn::CDynLib::FreeParameters
```

Definition at line 227 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.13 FreeParameterValue

```
helper::DynFuncHolder< decltype(bigvoice_FreeParameterValue) > sound4::bigvoice::dyn::CDynLib::FreeParameterValue
```

Definition at line 230 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.14 GetAudio

```
helper::DynFuncHolder< decltype(bigvoice_GetAudio) > sound4::bigvoice::dyn::CDynLib::GetAudio
```

Definition at line 250 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.15 GetAudioInputCount

```
helper::DynFuncHolder< decltype(bigvoice_GetAudioInputCount) > sound4::bigvoice::dyn::CDynLib::GetAudioInputCount
```

Definition at line 221 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.16 GetAudioOutputCount

```
helper::DynFuncHolder< decltype(bigvoice_GetAudioOutputCount) > sound4::bigvoice::dyn::CDynLib::GetAudioOutputCount
```

Definition at line 222 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.17 GetBufferIn

```
helper::DynFuncHolder< decltype(bigvoice_GetBufferIn) > sound4::bigvoice::dyn::CDynLib::GetBufferIn
```

Definition at line 240 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.18 GetBufferOut

```
helper::DynFuncHolder< decltype(bigvoice_GetBufferOut) > sound4::bigvoice::dyn::CDynLib::↔  
GetBufferOut
```

Definition at line 241 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.19 GetBytesFromFormat

```
helper::DynFuncHolder< decltype(bigvoice_GetBytesFromFormat) > sound4::bigvoice::dyn::CDyn↔  
Lib::GetBytesFromFormat
```

Definition at line 245 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.20 GetChannelCount

```
helper::DynFuncHolder< decltype(bigvoice_GetChannelCount) > sound4::bigvoice::dyn::CDynLib↔  
::GetChannelCount
```

Definition at line 220 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.21 GetChunkSizeInFrames

```
helper::DynFuncHolder< decltype(bigvoice_GetChunkSizeInFrames) > sound4::bigvoice::dyn::↔  
CDynLib::GetChunkSizeInFrames
```

Definition at line 219 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.22 GetEstimatedDelay

```
helper::DynFuncHolder< decltype(bigvoice_GetEstimatedDelay) > sound4::bigvoice::dyn::CDyn↔  
Lib::GetEstimatedDelay
```

Definition at line 242 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.23 GetFormatFromName

```
helper::DynFuncHolder< decltype(bigvoice_GetFormatFromName) > sound4::bigvoice::dyn::CDyn↔  
Lib::GetFormatFromName
```

Definition at line 244 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.24 GetFormatName

```
helper::DynFuncHolder< decltype(bigvoice_GetFormatName) > sound4::bigvoice::dyn::CDynLib::↵  
GetFormatName
```

Definition at line 243 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.25 GetMaxPacketFrame

```
helper::DynFuncHolder< decltype(bigvoice_GetMaxPacketFrame) > sound4::bigvoice::dyn::CDyn↵  
Lib::GetMaxPacketFrame
```

Definition at line 246 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.26 GetMetadataInfos

```
helper::DynFuncHolder< decltype(bigvoice_GetMetadataInfos) > sound4::bigvoice::dyn::CDynLib↵  
::GetMetadataInfos
```

Definition at line 276 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.27 GetOutputCount

```
helper::DynFuncHolder< decltype(bigvoice_GetOutputCount) > sound4::bigvoice::dyn::CDynLib::↵  
GetOutputCount
```

Definition at line 249 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.28 GetParameter

```
helper::DynFuncHolder< decltype(bigvoice_GetParameter) > sound4::bigvoice::dyn::CDynLib::↵  
GetParameter
```

Definition at line 229 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.29 GetPossibleChunkSizeInFrames

```
helper::DynFuncHolder< decltype(bigvoice_GetPossibleChunkSizeInFrames) > sound4::bigvoice↵  
::dyn::CDynLib::GetPossibleChunkSizeInFrames
```

Definition at line 284 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.30 GetProcessChunkFrames

```
helper::DynFuncHolder< decltype(bigvoice_GetProcessChunkFrames) > sound4::bigvoice::dyn::↔  
CDynLib::GetProcessChunkFrames
```

Definition at line 285 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.31 GetSampleRate

```
helper::DynFuncHolder< decltype(bigvoice_GetSampleRate) > sound4::bigvoice::dyn::CDynLib::↔  
GetSampleRate
```

Definition at line 223 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.32 GetVersion

```
helper::DynFuncHolder< decltype(bigvoice_GetVersion) > sound4::bigvoice::dyn::CDynLib::Get↔  
Version
```

Definition at line 218 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.33 InitProcess

```
helper::DynFuncHolder< decltype(bigvoice_InitProcess) > sound4::bigvoice::dyn::CDynLib::↔  
InitProcess
```

Definition at line 231 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.34 InitProcess2

```
helper::DynFuncHolder< decltype(bigvoice_InitProcess2) > sound4::bigvoice::dyn::CDynLib::↔  
InitProcess2
```

Definition at line 232 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.35 InitProcess3

```
helper::DynFuncHolder< decltype(bigvoice_InitProcess3) > sound4::bigvoice::dyn::CDynLib::↔  
InitProcess3
```

Definition at line 286 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.36 MonoToStereo

```
helper::DynFuncHolder< decltype(bigvoice_MonoToStereo) > sound4::bigvoice::dyn::CDynLib::MonoToStereo
```

Definition at line 254 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.37 MonoToStereo_Planar

```
helper::DynFuncHolder< decltype(bigvoice_MonoToStereo_Planar) > sound4::bigvoice::dyn::CDynLib::MonoToStereo_Planar
```

Definition at line 272 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.38 NewBus

```
helper::DynFuncHolder< decltype(bigvoice_NewBus) > sound4::bigvoice::dyn::CDynLib::NewBus
```

Definition at line 290 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.39 NewClient

```
helper::DynFuncHolder< decltype(bigvoice_NewClient) > sound4::bigvoice::dyn::CDynLib::NewClient
```

Definition at line 257 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.40 NewParameters

```
helper::DynFuncHolder< decltype(bigvoice_NewParameters) > sound4::bigvoice::dyn::CDynLib::NewParameters
```

Definition at line 226 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.41 PresetManager_InformChange

```
helper::DynFuncHolder< decltype(bigvoice_PresetManager_InformChange) > sound4::bigvoice::dyn::CDynLib::PresetManager_InformChange
```

Definition at line 281 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.42 ProcessAudio

```
helper::DynFuncHolder< decltype(bigvoice_ProcessAudio) > sound4::bigvoice::dyn::CDynLib::↔  
ProcessAudio
```

Definition at line 238 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.43 ProcessAudio_Planar

```
helper::DynFuncHolder< decltype(bigvoice_ProcessAudio_Planar) > sound4::bigvoice::dyn::CDyn↔  
Lib::ProcessAudio_Planar
```

Definition at line 239 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.44 ProcessJson

```
helper::DynFuncHolder< decltype(bigvoice_ProcessJson) > sound4::bigvoice::dyn::CDynLib::↔  
ProcessJson
```

Definition at line 259 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.45 SaveState

```
helper::DynFuncHolder< decltype(bigvoice_SaveState) > sound4::bigvoice::dyn::CDynLib::Save↔  
State
```

Definition at line 261 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.46 SetInstanceBus

```
helper::DynFuncHolder< decltype(bigvoice_SetInstanceBus) > sound4::bigvoice::dyn::CDynLib::↔  
SetInstanceBus
```

Definition at line 292 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.47 SetLoggerCallback

```
helper::DynFuncHolder< decltype(bigvoice_SetLoggerCallback) > sound4::bigvoice::dyn::CDyn↔  
Lib::SetLoggerCallback
```

Definition at line 224 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.48 SetLogSeverity

```
helper::DynFuncHolder< decltype(bigvoice_SetLogSeverity) > sound4::bigvoice::dyn::CDynLib::↵  
SetLogSeverity
```

Definition at line 225 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.49 SetMetadata

```
helper::DynFuncHolder< decltype(bigvoice_SetMetadata) > sound4::bigvoice::dyn::CDynLib::Set↵  
Metadata
```

Definition at line 275 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.50 SetParameter

```
helper::DynFuncHolder< decltype(bigvoice_SetParameter) > sound4::bigvoice::dyn::CDynLib::↵  
SetParameter
```

Definition at line 228 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.51 SetPresetManager

```
helper::DynFuncHolder< decltype(bigvoice_SetPresetManager) > sound4::bigvoice::dyn::CDynLib↵  
::SetPresetManager
```

Definition at line 280 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.52 StartUpdateThread

```
helper::DynFuncHolder< decltype(bigvoice_StartUpdateThread) > sound4::bigvoice::dyn::CDyn↵  
Lib::StartUpdateThread
```

Definition at line 235 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.53 StereoToMono

```
helper::DynFuncHolder< decltype(bigvoice_StereoToMono) > sound4::bigvoice::dyn::CDynLib::↵  
StereoToMono
```

Definition at line 253 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.54 StereoToMono_Planar

```
helper::DynFuncHolder< decltype(bigvoice_StereoToMono_Planar) > sound4::bigvoice::dyn::CDynLib↔  
Lib::StereoToMono_Planar
```

Definition at line 271 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.55 StopUpdateThread

```
helper::DynFuncHolder< decltype(bigvoice_StopUpdateThread) > sound4::bigvoice::dyn::CDynLib↔  
::StopUpdateThread
```

Definition at line 236 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.56 TerminateProcess

```
helper::DynFuncHolder< decltype(bigvoice_TerminateProcess) > sound4::bigvoice::dyn::CDynLib↔  
::TerminateProcess
```

Definition at line 233 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.57 WaitUpdateThreadReady

```
helper::DynFuncHolder< decltype(bigvoice_WaitUpdateThreadReady) > sound4::bigvoice::dyn::↔  
CDynLib::WaitUpdateThreadReady
```

Definition at line 237 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.58 Webserver

```
helper::DynFuncHolder< decltype(bigvoice_Webserver) > sound4::bigvoice::dyn::CDynLib::↔  
Webserver
```

Definition at line 265 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.59 Webserver_Status

```
helper::DynFuncHolder< decltype(bigvoice_Webserver_Status) > sound4::bigvoice::dyn::CDynLib↔  
::Webserver_Status
```

Definition at line 267 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.60 Webserver_Stop

```
helper::DynFuncHolder< decltype(bigvoice_Webserver_Stop) > sound4::bigvoice::dyn::CDynLib::↔
Webserver_Stop
```

Definition at line 266 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.61 Webserver_tcp

```
helper::DynFuncHolder< decltype(bigvoice_Webserver_tcp) > sound4::bigvoice::dyn::CDynLib::↔
Webserver_tcp
```

Definition at line 263 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.9.4.62 Webserver_tcp2

```
helper::DynFuncHolder< decltype(bigvoice_Webserver_tcp2) > sound4::bigvoice::dyn::CDynLib::↔
Webserver_tcp2
```

Definition at line 264 of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.10 sound4::bigvoice::dyn::helper::CDynLoader Class Reference

Helper to load dynamic library.

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Public Member Functions

- [CDynLoader](#) ()=default
- [CDynLoader](#) ([CDynLoader](#) &&)=default
- [CDynLoader](#) (const [CDynLoader](#) &)=delete
- [CDynLoader](#) & [operator=](#) ([CDynLoader](#) const &)=delete
- [operator bool](#) () const
- bool [isOk](#) () const
- bool [Load](#) (const std::filesystem::path &dynlib, uint32_t loadflags=LOAD_LIBRARY_SEARCH_DLL_LOAD↔
_DIR|LOAD_LIBRARY_SEARCH_APPLICATION_DIR|LOAD_LIBRARY_SEARCH_USER_DIRS|LOAD_↔
LIBRARY_SEARCH_SYSTEM32)
- void [Close](#) ()
- template<typename T>
[DynFuncHolder](#)< T > [GetSymbol](#) (const std::string &name)
- template<typename T>
[DynFuncHolder](#)< T > [GetSymbol_safe](#) (const std::string &name)
- virtual ~[CDynLoader](#) ()

Static Public Member Functions

- static std::filesystem::path [GetThisLibraryPath](#) (void)

10.10.1 Detailed Description

Definition at line [101](#) of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.2 Constructor & Destructor Documentation

10.10.2.1 CDynLoader() [1/3]

```
sound4::bigvoice::dyn::helper::CDynLoader::CDynLoader ( ) [default]
```

10.10.2.2 CDynLoader() [2/3]

```
sound4::bigvoice::dyn::helper::CDynLoader::CDynLoader (
    CDynLoader && ) [default]
```

10.10.2.3 CDynLoader() [3/3]

```
sound4::bigvoice::dyn::helper::CDynLoader::CDynLoader (
    const CDynLoader & ) [delete]
```

10.10.2.4 ~CDynLoader()

```
virtual sound4::bigvoice::dyn::helper::CDynLoader::~~CDynLoader ( ) [inline], [virtual]
```

Definition at line [198](#) of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.3 Member Function Documentation

10.10.3.1 Close()

```
void sound4::bigvoice::dyn::helper::CDynLoader::Close ( ) [inline]
```

Definition at line 159 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.3.2 GetSymbol()

```
template<typename T >  
DynFuncHolder<T> sound4::bigvoice::dyn::helper::CDynLoader::GetSymbol (   
    const std::string & name ) [inline]
```

Definition at line 172 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.3.3 GetSymbol_safe()

```
template<typename T >  
DynFuncHolder<T> sound4::bigvoice::dyn::helper::CDynLoader::GetSymbol_safe (   
    const std::string & name ) [inline]
```

Definition at line 180 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.3.4 GetThisLibraryPath()

```
static std::filesystem::path sound4::bigvoice::dyn::helper::CDynLoader::GetThisLibraryPath (   
    void ) [inline], [static]
```

Definition at line 184 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.3.5 IsOk()

```
bool sound4::bigvoice::dyn::helper::CDynLoader::IsOk ( ) const [inline]
```

Definition at line 113 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.3.6 Load()

```
bool sound4::bigvoice::dyn::helper::CDynLoader::Load (
    const std::filesystem::path & dynlib,
    uint32_t loadflags = LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR|LOAD_LIBRARY_SEARCH_APPLICATION_↵
    _DIR|LOAD_LIBRARY_SEARCH_USER_DIRS|LOAD_LIBRARY_SEARCH_SYSTEM32 ) [inline]
```

Definition at line 153 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.3.7 operator bool()

```
sound4::bigvoice::dyn::helper::CDynLoader::operator bool ( ) const [inline]
```

Definition at line 110 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.10.3.8 operator=()

```
CDynLoader& sound4::bigvoice::dyn::helper::CDynLoader::operator= (
    CDynLoader const & ) [delete]
```

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.11 sound4::bigvoice::CInstance Class Reference

```
#include <sound4.bigvoice.cl.hpp>
```

Classes

- class [CClient](#)

Public Member Functions

- [CInstance](#) ()
- [CInstance](#) (const [CInstance](#) &)=delete
- [CInstance](#) & [operator=](#) (const [CInstance](#) &)=delete
- [~CInstance](#) ()
- bool [IsOk](#) () const
- void [SetParam](#) (const std::string &name, const std::string &value)
- std::string [GetParam](#) (const std::string &name)
- void [SetPresetManager](#) ([CPresetLoader](#) *preset_manager)
- void [SetBus](#) (const [CBus](#) &bus)
- bool [Create](#) (const std::string &LoginKey, const std::string &RadioName, const std::string &AWS_Access_ID, const std::string &AWS_Access_Secret, const std::filesystem::path &save_path, int json_port=0, unsigned int frames_per_chunk=12)
- unsigned int [GetChunkFrames](#) ()
- void [PresetManager_InformChange](#) (const std::filesystem::path &relative_path, [PresetChange_Kind](#) change_kind)
- void [SetMetadata](#) (const std::string &key, const char *value)
- std::vector< std::tuple< std::string, std::string > > [GetMetadataInfos](#) ()
- virtual void [OnUpdateThreadStart](#) ()
- virtual void [OnUpdateThreadStop](#) ()
- bool [StartWebServer](#) (int http_port, int https_port=0)
- void [StopWebServer](#) ()
- int [Terminate](#) ()
- void [Stop](#) ()
- unsigned int [GetEstimatedDelay](#) ()
- std::array< float, [InputSampleSize](#) > & [GetBufferIn](#) ()
- std::array< float, [OutputSampleSize](#) > & [GetBufferOut](#) ()
- void [ProcessAudio](#) ()
- void [ProcessAudio_Planar](#) (float const *const *input, float *const *output)
- template<typename T >
std::vector< T > [ProcessAnyAudio](#) (const std::vector< T > input)
- std::shared_ptr< [CClient](#) > [NewClient](#) ()

Protected Member Functions

- template<typename T >
unsigned int [AddAudio](#) (const T *payload, unsigned int nFrame)
- template<typename T >
unsigned int [GetAudio](#) (T *payload, unsigned int max_nFrame)

10.11.1 Detailed Description

Instance handling class

Definition at line 385 of file [sound4.bigvoice.cl.hpp](#).

10.11.2 Constructor & Destructor Documentation

10.11.2.1 CInstance() [1/2]

```
sound4::bigvoice::CInstance::CInstance ( ) [inline]
```

Creates the instance, with default parameters

Definition at line 390 of file [sound4.bigvoice.cl.hpp](#).

10.11.2.2 CInstance() [2/2]

```
sound4::bigvoice::CInstance::CInstance (
    const CInstance & ) [delete]
```

10.11.2.3 ~CInstance()

```
sound4::bigvoice::CInstance::~~CInstance ( ) [inline]
```

Destructor, stopping any running tasks

Definition at line 401 of file [sound4.bigvoice.cl.hpp](#).

10.11.3 Member Function Documentation

10.11.3.1 AddAudio()

```
template<typename T >
unsigned int sound4::bigvoice::CInstance::AddAudio (
    const T * payload,
    unsigned int nFrame ) [inline], [protected]
```

Definition at line 826 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.2 Create()

```
bool sound4::bigvoice::CInstance::Create (
    const std::string & LoginKey,
    const std::string & RadioName,
    const std::string & AWS_Access_ID,
    const std::string & AWS_Access_Secret,
    const std::filesystem::path & save_path,
    int json_port = 0,
    unsigned int frames_per_chunk = 12 ) [inline]
```

Creates and initialize the instance, and starts its update thread

See also

[bigvoice_InitProcess2](#)

Parameters

<i>LoginKey</i>	SOUND4 Login key
<i>RadioName</i>	SOUND4 RadioName
<i>AWS_Access_ID</i>	SOUND4 AWS access ID
<i>AWS_Access_Secret</i>	SOUND4 AWS access Secret
<i>save_path</i>	Path where we can load/save the configuration
<i>json_port</i>	Optional json listening port
<i>frames_per_chunk</i>	The number of audio frames in each chunk of processing. Must be in the list returned by bigvoice_GetPossibleChunkSizeInFrames , or 0 to set default

Returns

false if failed

Definition at line 487 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.3 GetAudio()

```
template<typename T >
unsigned int sound4::bigvoice::CInstance::GetAudio (
    T * payload,
    unsigned int max_nFrame ) [inline], [protected]
```

Definition at line 832 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.4 GetBufferIn()

```
std::array<float, InputSampleSize>& sound4::bigvoice::CInstance::GetBufferIn ( ) [inline]
```

Gets the input buffer used in [ProcessAudio](#)

Input has GetAudioInputCount() * GetChannelCount() channels

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are interlaced, so the buffers are [L0,R0,L1,R1,...,Ln,Rn] (L=Left, R=Right)

In case of multiple audio input/output, buffer is [aL0,aR0,bL0,bR0,...,aLn,aRn,bLn,bRn]

Definition at line 681 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.5 GetBufferOut()

```
std::array<float, OutputSampleSize>& sound4::bigvoice::CInstance::GetBufferOut ( ) [inline]
```

Gets the output buffer used in [ProcessAudio](#)

Output has `GetAudioOutputCount() * GetChannelCount()` channels

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are interlaced, so the buffers are `[L0,R0,L1,R1,...,Ln,Rn]` (L=Left, R=Right)

In case of multiple audio input/output, buffer is `[aL0,aR0,bL0,bR0,...,aLn,aRn,bLn,bRn]`

Definition at line 698 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.6 GetChunkFrames()

```
unsigned int sound4::bigvoice::CInstance::GetChunkFrames ( ) [inline]
```

Get the number of audio frames in each chunk of processing.

The value reflects the `frames_per_chunk` parameter passed to [Create](#)

Returns

the number of audio frames in each chunk of processing

Definition at line 524 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.7 GetEstimatedDelay()

```
unsigned int sound4::bigvoice::CInstance::GetEstimatedDelay ( ) [inline]
```

Gets an estimated delay of current processing

See also

[bigvoice_GetEstimatedDelay](#)

Definition at line 665 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.8 GetMetadataInfos()

```
std::vector< std::tuple<std::string, std::string> > sound4::bigvoice::CInstance::GetMetadata↵  
Infos ( ) [inline]
```

Get information on known metadata keys.

Returns

a list of {key,description} of known metadata used in process and extensions.

Definition at line 568 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.9 GetParam()

```
std::string sound4::bigvoice::CInstance::GetParam (↵  
const std::string & name ) [inline]
```

Gets a parameter value

See also

[bigvoice_GetParameter](#)

Definition at line 427 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.10 IsOk()

```
bool sound4::bigvoice::CInstance::IsOk ( ) const [inline]
```

Returns true if an instance has been created and not stopped.

Definition at line 409 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.11 NewClient()

```
std::shared_ptr<CClient> sound4::bigvoice::CInstance::NewClient ( ) [inline]
```

Creates a low-level JSON client

See also

[JSON syntax](#)

Returns

a new client to communicate with the processing

Definition at line 818 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.12 OnUpdateThreadStart()

```
virtual void sound4::bigvoice::CInstance::OnUpdateThreadStart ( ) [inline], [virtual]
```

Definition at line 588 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.13 OnUpdateThreadStop()

```
virtual void sound4::bigvoice::CInstance::OnUpdateThreadStop ( ) [inline], [virtual]
```

Definition at line 590 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.14 operator=()

```
CInstance& sound4::bigvoice::CInstance::operator= (
    const CInstance & ) [delete]
```

10.11.3.15 PresetManager_InformChange()

```
void sound4::bigvoice::CInstance::PresetManager_InformChange (
    const std::filesystem::path & relative_path,
    PresetChange_Kind change_kind ) [inline]
```

Inform that a change happened in the storage. This must be used only in conjunction with [SetPresetManager](#)

Parameters

<i>relative_path</i>	The path which changed, relative to the root storage
<i>change_kind</i>	What happened to this path

Definition at line 538 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.16 ProcessAnyAudio()

```
template<typename T >
std::vector<T> sound4::bigvoice::CInstance::ProcessAnyAudio (
    const std::vector< T > input ) [inline]
```

Process audio (different formats, less optimized)

Supports for int16_t, int32_t and float (32) in native endianness

Attention

the return count is independent from the input count due to BIGVOICE_AUDIOFRAME_COUNT

Template Parameters

<i>T</i>	the C type for one audio sample
----------	---------------------------------

Parameters

<i>input</i>	audio to process
--------------	------------------

Returns

processed audio

Definition at line 746 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.17 ProcessAudio()

```
void sound4::bigvoice::CInstance::ProcessAudio ( ) [inline]
```

Process one audio chunk, from [GetBufferIn\(\)](#) to [GetBufferOut\(\)](#)

See also

[bigvoice_ProcessAudio](#)

Definition at line 709 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.18 ProcessAudio_Planar()

```
void sound4::bigvoice::CInstance::ProcessAudio_Planar (
    float const *const * input,
    float *const * output ) [inline]
```

Process one audio chunk, but with planar audio

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are NOT interlaced, so the buffers are arrays of planar audio : [[L0,L1,...,Ln] , [R0,R1,...,Rn]] (L=Left, R=Right)

In case of multiple audio input/output, buffer are [[aL...], [aR...], [bL...], [bR...], ...]

See also

[bigvoice_ProcessAudio_Planar](#)

Parameters

<i>input</i>	has GetAudioInputCount() * GetChannelCount() channels
<i>output</i>	has GetAudioOutputCount() * GetChannelCount() channels

Definition at line 728 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.19 SetBus()

```
void sound4::bigvoice::CInstance::SetBus (
    const CBus & bus ) [inline]
```

Set the instance bus

Definition at line 467 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.20 SetMetadata()

```
void sound4::bigvoice::CInstance::SetMetadata (
    const std::string & key,
    const char * value ) [inline]
```

Adds the specific metadata to the process.

It can come from any thread, and will be used as soon as possible. Setting a null value erase the metadata, which is different from setting an empty string.

Parameters

<i>key</i>	the metadata key If key contains a dot (like wm.KEY), it will be sent only to the matching plugin (ie wm here) without the prefix (ie KEY here) Use empty prefix for routing to main only.
<i>value</i>	the metadata value or nullptr to erase it.

Definition at line 554 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.21 SetParam()

```
void sound4::bigvoice::CInstance::SetParam (
    const std::string & name,
    const std::string & value ) [inline]
```

Set a parameter value

Must be done before [Create](#)

See also

[bigvoice_SetParameter](#)

Definition at line 417 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.22 SetPresetManager()

```
void sound4::bigvoice::CInstance::SetPresetManager (
    CPresetLoader * preset_manager ) [inline]
```

Sets a custom preset manager Optional, this overrides the `save_path` in [Create](#)

See also

[bigvoice_SetPresetManager](#)

Must be done before [Create](#)

Parameters

<i>preset_manager</i>	The custom preset manager
-----------------------	---------------------------

Definition at line 445 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.23 StartWebServer()

```
bool sound4::bigvoice::CInstance::StartWebServer (
    int http_port,
    int https_port = 0 ) [inline]
```

Starts a Web server on requested ports

See also

[bigvoice_Webserver](#)

Definition at line 598 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.24 Stop()

```
void sound4::bigvoice::CInstance::Stop ( ) [inline]
```

Stops the instance and its web server and its update thread.

See also

[bigvoice_ExitProcess](#),
[bigvoice_StopUpdateThread](#) and
[bigvoice_Webserver_Stop](#)

Definition at line 646 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.25 StopWebServer()

```
void sound4::bigvoice::CInstance::StopWebServer ( ) [inline]
```

Stops the Web server

See also

[bigvoice_Webserver_Stop](#)

Definition at line 620 of file [sound4.bigvoice.cl.hpp](#).

10.11.3.26 Terminate()

```
int sound4::bigvoice::CInstance::Terminate ( ) [inline]
```

Terminate the processing. This is useful only with file processing, to get the final status.

See also

[bigvoice_TerminateProcess](#)

Definition at line 635 of file [sound4.bigvoice.cl.hpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl.hpp](#)

10.12 sound4::bigvoice::dyn::CInstance Class Reference

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Classes

- class [CClient](#)

Public Member Functions

- [CInstance](#) ([CDynLib](#) &dynlib)
- [CInstance](#) (const [CInstance](#) &)=delete
- [CInstance](#) & [operator=](#) (const [CInstance](#) &)=delete
- [~CInstance](#) ()
- bool [IsOk](#) () const
- void [SetParam](#) (const std::string &name, const std::string &value)
- std::string [GetParam](#) (const std::string &name)
- void [SetBus](#) (const [CBus](#) &bus)
- void [SetPresetManager](#) ([CPresetLoader](#) *preset_manager)
- bool [Create](#) (const std::string &LoginKey, const std::string &RadioName, const std::string &AWS_Access_ID, const std::string &AWS_Access_Secret, const std::filesystem::path &save_path, int json_port=0, unsigned int frames_per_chunk=12)
- unsigned int [GetChunkFrames](#) ()
- void [PresetManager_InformChange](#) (const std::filesystem::path &relative_path, [PresetChange_Kind](#) change_kind)
- void [SetMetadata](#) (const std::string &key, const char *value)
- std::vector< std::tuple< std::string, std::string > > [GetMetadataInfos](#) ()
- virtual void [OnUpdateThreadStart](#) ()
- virtual void [OnUpdateThreadStop](#) ()
- bool [StartWebServer](#) (int http_port, int https_port=0)
- void [StopWebServer](#) ()
- int [Terminate](#) ()
- void [Stop](#) ()
- unsigned int [GetEstimatedDelay](#) ()
- std::array< float, [InputSampleSize](#) > & [GetBufferIn](#) ()
- std::array< float, [OutputSampleSize](#) > & [GetBufferOut](#) ()
- void [ProcessAudio](#) ()
- void [ProcessAudio_Planar](#) (float const *const *input, float *const *output)
- template<typename T >
std::vector< T > [ProcessAnyAudio](#) (const std::vector< T > input)
- std::shared_ptr< [CClient](#) > [NewClient](#) ()

Protected Member Functions

- template<typename T >
unsigned int [AddAudio](#) (const T *payload, unsigned int nFrame)
- template<typename T >
unsigned int [GetAudio](#) (T *payload, unsigned int max_nFrame)

10.12.1 Detailed Description

Instance handling class

Definition at line 770 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.2 Constructor & Destructor Documentation

10.12.2.1 CInstance() [1/2]

```
sound4::bigvoice::dyn::CInstance::CInstance (
    CDynLib & dynlib ) [inline]
```

Creates the instance, with default parameters

Definition at line 775 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.2.2 CInstance() [2/2]

```
sound4::bigvoice::dyn::CInstance::CInstance (
    const CInstance & ) [delete]
```

10.12.2.3 ~CInstance()

```
sound4::bigvoice::dyn::CInstance::~~CInstance ( ) [inline]
```

Destructor, stopping any running tasks

Definition at line 787 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3 Member Function Documentation

10.12.3.1 AddAudio()

```
template<typename T >
unsigned int sound4::bigvoice::dyn::CInstance::AddAudio (
    const T * payload,
    unsigned int nFrame ) [inline], [protected]
```

Definition at line 1228 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.2 Create()

```
bool sound4::bigvoice::dyn::CInstance::Create (
    const std::string & LoginKey,
    const std::string & RadioName,
    const std::string & AWS_Access_ID,
    const std::string & AWS_Access_Secret,
    const std::filesystem::path & save_path,
    int json_port = 0,
    unsigned int frames_per_chunk = 12 ) [inline]
```

Creates and initialize the instance, and starts its update thread

See also

[bigvoice_InitProcess2](#)

Parameters

<i>LoginKey</i>	SOUND4 Login key
<i>RadioName</i>	SOUND4 RadioName
<i>AWS_Access_ID</i>	SOUND4 AWS access ID
<i>AWS_Access_Secret</i>	SOUND4 AWS access Secret
<i>save_path</i>	Path where we can load/save the configuration
<i>json_port</i>	Optional json listening port
<i>frames_per_chunk</i>	The number of audio frames in each chunk of processing. Must be in the list returned by bigvoice_GetPossibleChunkSizeInFrames , or 0 to set default

Returns

false if failed

Definition at line 875 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.3 GetAudio()

```
template<typename T >
unsigned int sound4::bigvoice::dyn::CInstance::GetAudio (
    T * payload,
    unsigned int max_nFrame ) [inline], [protected]
```

Definition at line 1234 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.4 GetBufferIn()

```
std::array<float, InputSampleSize>& sound4::bigvoice::dyn::CInstance::GetBufferIn ( ) [inline]
```

Gets the input buffer used in [ProcessAudio](#)

Input has GetAudioInputCount() * GetChannelCount() channels

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are interlaced, so the buffers are [L0,R0,L1,R1,...,Ln,Rn] (L=Left, R=Right)

In case of multiple audio input/output, buffer is [aL0,aR0,bL0,bR0,...,aLn,aRn,bLn,bRn]

Definition at line 1081 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.5 GetBufferOut()

```
std::array<float, OutputSampleSize>& sound4::bigvoice::dyn::CInstance::GetBufferOut ( ) [inline]
```

Gets the output buffer used in [ProcessAudio](#)

Output has `GetAudioOutputCount() * GetChannelCount()` channels

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are interlaced, so the buffers are `[L0,R0,L1,R1,...,Ln,Rn]` (L=Left, R=Right)

In case of multiple audio input/output, buffer is `[aL0,aR0,bL0,bR0,...,aLn,aRn,bLn,bRn]`

Definition at line 1098 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.6 GetChunkFrames()

```
unsigned int sound4::bigvoice::dyn::CInstance::GetChunkFrames ( ) [inline]
```

Get the number of audio frames in each chunk of processing.

The value reflects the `frames_per_chunk` parameter passed to [Create](#)

Returns

the number of audio frames in each chunk of processing

Definition at line 918 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.7 GetEstimatedDelay()

```
unsigned int sound4::bigvoice::dyn::CInstance::GetEstimatedDelay ( ) [inline]
```

Gets an estimated delay of current processing

See also

[bigvoice_GetEstimatedDelay](#)

Definition at line 1065 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.8 GetMetadataInfos()

```
std::vector< std::tuple<std::string, std::string> > sound4::bigvoice::dyn::CInstance::GetMetadataInfos ( ) [inline]
```

Get information on known metadata keys.

Returns

a list of {key,description} of known metadata used in process and extensions.

Definition at line 966 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.9 GetParam()

```
std::string sound4::bigvoice::dyn::CInstance::GetParam (
    const std::string & name ) [inline]
```

Gets a parameter value

See also

[bigvoice_GetParameter](#)

Definition at line 813 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.10 IsOk()

```
bool sound4::bigvoice::dyn::CInstance::IsOk ( ) const [inline]
```

Returns true if an instance has been created and not stopped.

Definition at line 795 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.11 NewClient()

```
std::shared_ptr<CClient> sound4::bigvoice::dyn::CInstance::NewClient ( ) [inline]
```

Creates a low-level JSON client

See also

[JSON syntax](#)

Returns

a new client to communicate with the processing

Definition at line 1220 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.12 OnUpdateThreadStart()

```
virtual void sound4::bigvoice::dyn::CInstance::OnUpdateThreadStart ( ) [inline], [virtual]
```

Definition at line 988 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.13 OnUpdateThreadStop()

```
virtual void sound4::bigvoice::dyn::CInstance::OnUpdateThreadStop ( ) [inline], [virtual]
```

Definition at line 990 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.14 operator=()

```
CInstance& sound4::bigvoice::dyn::CInstance::operator= (
    const CInstance & ) [delete]
```

10.12.3.15 PresetManager_InformChange()

```
void sound4::bigvoice::dyn::CInstance::PresetManager_InformChange (
    const std::filesystem::path & relative_path,
    PresetChange_Kind change_kind ) [inline]
```

Inform that a change happened in the storage. This must be used only in conjunction with [SetPresetManager](#)

Parameters

<i>relative_path</i>	The path which changed, relative to the root storage
<i>change_kind</i>	What happened to this path

Definition at line 932 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.16 ProcessAnyAudio()

```
template<typename T >
std::vector<T> sound4::bigvoice::dyn::CInstance::ProcessAnyAudio (
    const std::vector< T > input ) [inline]
```

Process audio (different formats, less optimized)

Supports for int16_t, int32_t and float (32) in native endianness

Attention

the return count is independent from the input count due to BIGVOICE_AUDIOFRAME_COUNT

Template Parameters

<i>T</i>	the C type for one audio sample
----------	---------------------------------

Parameters

<i>input</i>	audio to process
--------------	------------------

Returns

processed audio

Definition at line 1146 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.17 ProcessAudio()

```
void sound4::bigvoice::dyn::CInstance::ProcessAudio ( ) [inline]
```

Process one audio chunk, from [GetBufferIn\(\)](#) to [GetBufferOut\(\)](#)

See also

[bigvoice_ProcessAudio](#)

Definition at line 1109 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.18 ProcessAudio_Planar()

```
void sound4::bigvoice::dyn::CInstance::ProcessAudio_Planar (
    float const *const * input,
    float *const * output ) [inline]
```

Process one audio chunk, but with planar audio

Each sample is a native 32-bit floating-point number where 1.0 is full scale. Channels are NOT interlaced, so the buffers are arrays of planar audio : [[L0,L1,...,Ln] , [R0,R1,...,Rn]] (L=Left, R=Right)

In case of multiple audio input/output, buffer are [[aL...], [aR...], [bL...], [bR...], ...]

See also

[bigvoice_ProcessAudio_Planar](#)

Parameters

<i>input</i>	has GetAudioInputCount() * GetChannelCount() channels
<i>output</i>	has GetAudioOutputCount() * GetChannelCount() channels

Definition at line 1128 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.19 SetBus()

```
void sound4::bigvoice::dyn::CInstance::SetBus (
    const CBus & bus ) [inline]
```

Set the instance bus

Definition at line 825 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.20 SetMetadata()

```
void sound4::bigvoice::dyn::CInstance::SetMetadata (
    const std::string & key,
    const char * value ) [inline]
```

Adds the specific metadata to the process.

It can come from any thread, and will be used as soon as possible. Setting a null value erase the metadata, which is different from setting an empty string.

Parameters

<i>key</i>	the metadata key If key contains a dot (like wm.KEY), it will be sent only to the matching plugin (ie wm here) without the prefix (ie KEY here) Use empty prefix for routing to main only.
<i>value</i>	the metadata value or nullptr to erase it.

Definition at line 950 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.21 SetParam()

```
void sound4::bigvoice::dyn::CInstance::SetParam (
    const std::string & name,
    const std::string & value ) [inline]
```

Set a parameter value

Must be done before [Create](#)

See also

[bigvoice_SetParameter](#)

Definition at line 803 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.22 SetPresetManager()

```
void sound4::bigvoice::dyn::CInstance::SetPresetManager (
    CPresetLoader * preset_manager ) [inline]
```

Sets a custom preset manager Optional, this overrides the `save_path` in [Create](#)

See also

[bigvoice_SetPresetManager](#)

Must be done before [Create](#)

Parameters

<i>preset_manager</i>	The custom preset manager
-----------------------	---------------------------

Definition at line 842 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.23 StartWebServer()

```
bool sound4::bigvoice::dyn::CInstance::StartWebServer (
    int http_port,
    int https_port = 0 ) [inline]
```

Starts a Web server on requested ports

See also

[bigvoice_Webserver](#)

Definition at line 998 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.24 Stop()

```
void sound4::bigvoice::dyn::CInstance::Stop ( ) [inline]
```

Stops the instance and its web server and its update thread.

See also

[bigvoice_ExitProcess](#),
[bigvoice_StopUpdateThread](#) and
[bigvoice_Webserver_Stop](#)

Definition at line 1046 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.25 StopWebServer()

```
void sound4::bigvoice::dyn::CInstance::StopWebServer ( ) [inline]
```

Stops the Web server

See also

[bigvoice_Webserver_Stop](#)

Definition at line 1020 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.12.3.26 Terminate()

```
int sound4::bigvoice::dyn::CInstance::Terminate ( ) [inline]
```

Terminate the processing. This is useful only with file processing, to get the final status.

See also

[bigvoice_TerminateProcess](#)

Definition at line 1035 of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.13 sound4::bigvoice::CPresetLoader Class Reference

```
#include <sound4.bigvoice.cl.hpp>
```

Public Member Functions

- [CPresetLoader](#) ()=default
- virtual [~CPresetLoader](#) ()=default
- virtual bool [IsReadOnly](#) ()=0
- virtual bool [Exists](#) (const std::filesystem::path &name)=0
- virtual bool [Remove](#) (const std::filesystem::path &name)=0
- virtual bool [Rename](#) (const std::filesystem::path &from, const std::filesystem::path &to)=0
- virtual std::vector< std::filesystem::path > [GetAll](#) ()=0
- virtual std::string [Read](#) (const std::filesystem::path &filename)=0
- virtual bool [Write](#) (const std::filesystem::path &filename, const std::string &content)=0

10.13.1 Detailed Description

Custom preset handler helper Derive from this class and call [CInstance::SetPresetManager](#)

Definition at line 367 of file [sound4.bigvoice.cl.hpp](#).

10.13.2 Constructor & Destructor Documentation

10.13.2.1 CPresetLoader()

```
sound4::bigvoice::CPresetLoader::CPresetLoader ( ) [default]
```

10.13.2.2 ~CPresetLoader()

```
virtual sound4::bigvoice::CPresetLoader::~~CPresetLoader ( ) [virtual], [default]
```

10.13.3 Member Function Documentation

10.13.3.1 Exists()

```
virtual bool sound4::bigvoice::CPresetLoader::Exists (
    const std::filesystem::path & name ) [pure virtual]
```

10.13.3.2 GetAll()

```
virtual std::vector<std::filesystem::path> sound4::bigvoice::CPresetLoader::GetAll ( ) [pure virtual]
```

10.13.3.3 IsReadOnly()

```
virtual bool sound4::bigvoice::CPresetLoader::IsReadOnly ( ) [pure virtual]
```

10.13.3.4 Read()

```
virtual std::string sound4::bigvoice::CPresetLoader::Read (
    const std::filesystem::path & filename ) [pure virtual]
```

10.13.3.5 Remove()

```
virtual bool sound4::bigvoice::CPresetLoader::Remove (
    const std::filesystem::path & name ) [pure virtual]
```

10.13.3.6 Rename()

```
virtual bool sound4::bigvoice::CPresetLoader::Rename (
    const std::filesystem::path & from,
    const std::filesystem::path & to ) [pure virtual]
```

10.13.3.7 Write()

```
virtual bool sound4::bigvoice::CPresetLoader::Write (
    const std::filesystem::path & filename,
    const std::string & content ) [pure virtual]
```

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl.hpp](#)

10.14 sound4::bigvoice::dyn::CPresetLoader Class Reference

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Inherited by [MyPresetManager](#).

Public Member Functions

- [CPresetLoader](#) ()=default
- virtual [~CPresetLoader](#) ()=default
- virtual bool [IsReadOnly](#) ()=0
- virtual bool [Exists](#) (const std::filesystem::path &name)=0
- virtual bool [Remove](#) (const std::filesystem::path &name)=0
- virtual bool [Rename](#) (const std::filesystem::path &from, const std::filesystem::path &to)=0
- virtual std::vector< std::filesystem::path > [GetAll](#) ()=0
- virtual std::string [Read](#) (const std::filesystem::path &filename)=0
- virtual bool [Write](#) (const std::filesystem::path &filename, const std::string &content)=0

10.14.1 Detailed Description

Custom preset handler helper Derive from this class and call [CInstance::SetPresetManager](#)

Definition at line 752 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.14.2 Constructor & Destructor Documentation

10.14.2.1 CPresetLoader()

```
sound4::bigvoice::dyn::CPresetLoader::CPresetLoader ( ) [default]
```

10.14.2.2 ~CPresetLoader()

```
virtual sound4::bigvoice::dyn::CPresetLoader::~~CPresetLoader ( ) [virtual], [default]
```

10.14.3 Member Function Documentation

10.14.3.1 Exists()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Exists (
    const std::filesystem::path & name ) [pure virtual]
```

Implemented in [MyPresetManager](#).

10.14.3.2 GetAll()

```
virtual std::vector<std::filesystem::path> sound4::bigvoice::dyn::CPresetLoader::GetAll ( )
[pure virtual]
```

Implemented in [MyPresetManager](#).

10.14.3.3 IsReadOnly()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::IsReadOnly ( ) [pure virtual]
```

Implemented in [MyPresetManager](#).

10.14.3.4 Read()

```
virtual std::string sound4::bigvoice::dyn::CPresetLoader::Read (
    const std::filesystem::path & filename ) [pure virtual]
```

Implemented in [MyPresetManager](#).

10.14.3.5 Remove()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Remove (
    const std::filesystem::path & name ) [pure virtual]
```

Implemented in [MyPresetManager](#).

10.14.3.6 Rename()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Rename (
    const std::filesystem::path & from,
    const std::filesystem::path & to ) [pure virtual]
```

Implemented in [MyPresetManager](#).

10.14.3.7 Write()

```
virtual bool sound4::bigvoice::dyn::CPresetLoader::Write (
    const std::filesystem::path & filename,
    const std::string & content ) [pure virtual]
```

Implemented in [MyPresetManager](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.15 sound4::bigvoice::dyn::helper::DynFuncHolder< T > Class Template Reference

Helper to wrap a function pointer.

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Public Member Functions

- [DynFuncHolder](#) ()=default
- [DynFuncHolder](#) ([DynFunc_t](#) a_ptr)
- [operator bool](#) () const
- [bool](#) [isOk](#) () const
- [operator T*](#) () const

10.15.1 Detailed Description

```
template<typename T>
class sound4::bigvoice::dyn::helper::DynFuncHolder< T >
```

Parameters

T	is the function type, you may use decltype(fn)
-------------------	--

Definition at line 86 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.15.2 Constructor & Destructor Documentation

10.15.2.1 DynFuncHolder() [1/2]

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::DynFuncHolder ( ) [default]
```

10.15.2.2 DynFuncHolder() [2/2]

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::DynFuncHolder (
    DynFunc_t a_ptr ) [inline]
```

Definition at line 91 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.15.3 Member Function Documentation

10.15.3.1 IsOk()

```
template<typename T >
bool sound4::bigvoice::dyn::helper::DynFuncHolder< T >::IsOk ( ) const [inline]
```

Definition at line 93 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.15.3.2 operator bool()

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::operator bool ( ) const [inline]
```

Definition at line 92 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.15.3.3 operator T*()

```
template<typename T >
sound4::bigvoice::dyn::helper::DynFuncHolder< T >::operator T* ( ) const [inline]
```

Definition at line 94 of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.16 MyPresetManager Class Reference

[Custom preset manager]

Inherits [sound4::bigvoice::dyn::CPresetLoader](#).

Public Member Functions

- [MyPresetManager](#) ()=default
- virtual [~MyPresetManager](#) ()=default
- bool [SetupPath](#) (const std::filesystem::path &a_preset_path)
- virtual bool [IsReadOnly](#) () override
- virtual bool [Exists](#) (const std::filesystem::path &name) override
- virtual bool [Remove](#) (const std::filesystem::path &name) override
- virtual bool [Rename](#) (const std::filesystem::path &from, const std::filesystem::path &to) override
- virtual std::vector< std::filesystem::path > [GetAll](#) () override
- virtual std::string [Read](#) (const std::filesystem::path &filename) override
- virtual bool [Write](#) (const std::filesystem::path &filename, const std::string &content) override

10.16.1 Detailed Description

Definition at line 119 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

10.16.2 Constructor & Destructor Documentation

10.16.2.1 MyPresetManager()

```
MyPresetManager::MyPresetManager ( ) [default]
```

10.16.2.2 ~MyPresetManager()

```
virtual MyPresetManager::~~MyPresetManager ( ) [virtual], [default]
```

10.16.3 Member Function Documentation

10.16.3.1 Exists()

```
virtual bool MyPresetManager::Exists (
    const std::filesystem::path & name ) [inline], [override], [virtual]
```

Implements [sound4::bigvoice::dyn::CPresetLoader](#).

Definition at line 137 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

10.16.3.2 GetAll()

```
virtual std::vector<std::filesystem::path> MyPresetManager::GetAll ( ) [inline], [override], [virtual]
```

Implements [sound4::bigvoice::dyn::CPresetLoader](#).

Definition at line 153 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

10.16.3.3 IsReadOnly()

```
virtual bool MyPresetManager::IsReadOnly ( ) [inline], [override], [virtual]
```

Implements [sound4::bigvoice::dyn::CPresetLoader](#).

Definition at line 136 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

10.16.3.4 Read()

```
virtual std::string MyPresetManager::Read (
    const std::filesystem::path & filename ) [inline], [override], [virtual]
```

Implements [sound4::bigvoice::dyn::CPresetLoader](#).

Definition at line 163 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

10.16.3.5 Remove()

```
virtual bool MyPresetManager::Remove (
    const std::filesystem::path & name ) [inline], [override], [virtual]
```

Implements [sound4::bigvoice::dyn::CPresetLoader](#).

Definition at line 142 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

10.16.3.6 Rename()

```
virtual bool MyPresetManager::Rename (
    const std::filesystem::path & from,
    const std::filesystem::path & to ) [inline], [override], [virtual]
```

Implements [sound4::bigvoice::dyn::CPresetLoader](#).

Definition at line 147 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

10.16.3.7 SetupPath()

```
bool MyPresetManager::SetupPath (
    const std::filesystem::path & a_preset_path ) [inline]
```

Definition at line 124 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

10.16.3.8 Write()

```
virtual bool MyPresetManager::Write (
    const std::filesystem::path & filename,
    const std::string & content ) [inline], [override], [virtual]
```

Implements [sound4::bigvoice::dyn::CPresetLoader](#).

Definition at line 173 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

The documentation for this class was generated from the following file:

- [sound4.bigvoice.cl-pipe-dyn.cpp](#)

10.17 sound4::bigvoice::dyn::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

10.17.1 Detailed Description

```
template<typename T>
struct sound4::bigvoice::dyn::helper::SampleFormat< T >
```

Template Parameters

<i>T</i>	the C type for one audio sample
----------	---------------------------------

Definition at line 489 of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.18 sound4::bigvoice::helper::SampleFormat< T > Struct Template Reference

Helper for Sample format types.

```
#include <sound4.bigvoice.cl.hpp>
```

10.18.1 Detailed Description

```
template<typename T>
struct sound4::bigvoice::helper::SampleFormat< T >
```

Template Parameters

<i>T</i>	the C type for one audio sample
----------	---------------------------------

Definition at line 116 of file [sound4.bigvoice.cl.hpp](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl.hpp](#)

10.19 sound4::bigvoice::dyn::helper::SampleFormat< float > Struct Reference

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Public Attributes

- const [bigvoice_SampleFormat](#) format = F32_NATIVE

10.19.1 Detailed Description

Definition at line 500 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.19.2 Member Data Documentation

10.19.2.1 format

```
const bigvoice_SampleFormat sound4::bigvoice::dyn::helper::SampleFormat< float >::format =  
F32_NATIVE
```

Definition at line 501 of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.20 sound4::bigvoice::helper::SampleFormat< float > Struct Reference

```
#include <sound4.bigvoice.cl.hpp>
```

Public Attributes

- const [bigvoice_SampleFormat](#) format = [F32_NATIVE](#)

10.20.1 Detailed Description

Definition at line 127 of file [sound4.bigvoice.cl.hpp](#).

10.20.2 Member Data Documentation

10.20.2.1 format

```
const bigvoice_SampleFormat sound4::bigvoice::helper::SampleFormat< float >::format = F32\_NATIVE
```

Definition at line 128 of file [sound4.bigvoice.cl.hpp](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl.hpp](#)

10.21 sound4::bigvoice::dyn::helper::SampleFormat< int16_t > Struct Reference

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Public Attributes

- const [bigvoice_SampleFormat](#) format = [S16_NATIVE](#)

10.21.1 Detailed Description

Definition at line [492](#) of file [sound4.bigvoice.cl_dyn.hpp](#).

10.21.2 Member Data Documentation

10.21.2.1 format

```
const bigvoice\_SampleFormat sound4::bigvoice::dyn::helper::SampleFormat< int16_t >::format =  
S16\_NATIVE
```

Definition at line [493](#) of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.22 [sound4::bigvoice::helper::SampleFormat< int16_t >](#) Struct Reference

```
#include <sound4.bigvoice.cl.hpp>
```

Public Attributes

- const [bigvoice_SampleFormat](#) format = [S16_NATIVE](#)

10.22.1 Detailed Description

Definition at line [119](#) of file [sound4.bigvoice.cl.hpp](#).

10.22.2 Member Data Documentation

10.22.2.1 format

```
const bigvoice_SampleFormat sound4::bigvoice::helper::SampleFormat< int16_t >::format = S16_NATIVE
```

Definition at line 120 of file [sound4.bigvoice.cl.hpp](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl.hpp](#)

10.23 sound4::bigvoice::dyn::helper::SampleFormat< int32_t > Struct Reference

```
#include <sound4.bigvoice.cl_dyn.hpp>
```

Public Attributes

- const [bigvoice_SampleFormat](#) format = S32_NATIVE

10.23.1 Detailed Description

Definition at line 496 of file [sound4.bigvoice.cl_dyn.hpp](#).

10.23.2 Member Data Documentation

10.23.2.1 format

```
const bigvoice_SampleFormat sound4::bigvoice::dyn::helper::SampleFormat< int32_t >::format = S32_NATIVE
```

Definition at line 497 of file [sound4.bigvoice.cl_dyn.hpp](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl_dyn.hpp](#)

10.24 sound4::bigvoice::helper::SampleFormat< int32_t > Struct Reference

```
#include <sound4.bigvoice.cl.hpp>
```

Public Attributes

- `const bigvoice\_SampleFormat format = S32\_NATIVE`

10.24.1 Detailed Description

Definition at line [123](#) of file [sound4.bigvoice.cl.hpp](#).

10.24.2 Member Data Documentation

10.24.2.1 `format`

```
const bigvoice\_SampleFormat sound4::bigvoice::helper::SampleFormat< int32\_t >::format = S32\_NATIVE
```

Definition at line [124](#) of file [sound4.bigvoice.cl.hpp](#).

The documentation for this struct was generated from the following file:

- [sound4.bigvoice.cl.hpp](#)

Chapter 11

File Documentation

11.1 documentation.h File Reference

11.2 documentation.h

00001

11.3 sound4.bigvoice.cl-pipe-cxx.cpp File Reference

C++ sample using C++ interface wrapper.

```
#include <unistd.h>
#include <iostream>
#include <strings.h>
#include <sys/types.h>
#include <filesystem>
#include <io.h>
#include <fcntl.h>
#include "sound4.bigvoice.cl.hpp"
```

Functions

- std::string [GetEnv](#) (const std::string &name)
- void [ProcessLoop](#) ([sound4::bigvoice::CInstance](#) &instance)
- void [Usage](#) (const char *progrname)
- void [MyLogger](#) ([sound4::bigvoice::LogSeverity](#) severity, const std::string &msg)
- int [main](#) (int argc, const char **argv)

11.3.1 Function Documentation

11.3.1.1 GetEnv()

```
std::string GetEnv (
    const std::string & name )
```

Definition at line 39 of file [sound4.bigvoice.cl-pipe-cxx.cpp](#).

11.3.1.2 main()

```
int main (
    int argc,
    const char ** argv )
```

[Setup and sanity checks]

[Setup and sanity checks]

[Create the processing instance]

[Create the processing instance]

[Communicating with the instance]

[Communicating with the instance]

Definition at line 104 of file [sound4.bigvoice.cl-pipe-cxx.cpp](#).

11.3.1.3 MyLogger()

```
void MyLogger (
    sound4::bigvoice::LogSeverity severity,
    const std::string & msg )
```

Definition at line 99 of file [sound4.bigvoice.cl-pipe-cxx.cpp](#).

11.3.1.4 ProcessLoop()

```
void ProcessLoop (
    sound4::bigvoice::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 48 of file [sound4.bigvoice.cl-pipe-cxx.cpp](#).

11.3.1.5 Usage()

```
void Usage (
    const char * progname )
```

Definition at line 79 of file [sound4.bigvoice.cl-pipe-cxx.cpp](#).

11.4 sound4.bigvoice.cl-pipe-cxx.cpp

```
00001
00004 #include <unistd.h>
00005 #include <iostream>
00006 #include <strings.h>
00007 #include <sys/types.h>
00008 #include <filesystem>
00009 #if defined(_WIN32) || defined(WIN32)
00010     #include <io.h>
00011     #include <fcntl.h>
00012 #endif
00013
00014 #include "sound4.bigvoice.cl.hpp"
00015
00016 // -----
00039 std::string GetEnv(const std::string& name)
00040 {
00041     char *data=getenv(name.c_str());
00042     if (!data) return {};
00043     return std::string(data);
00044 }
00045
00046
00047 // Reads from stdin, write to stdout
00048 void ProcessLoop(sound4::bigvoice::CInstance &instance)
00049 {
00050     bool stopthread=false;
00051
00052     #if defined(_WIN32) || defined(WIN32)
00053         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00054         _setmode(_fileno(stdin), _O_BINARY);
00055         _setmode(_fileno(stdout), _O_BINARY);
00056     #endif
00057
00058     while (!stopthread)
00059     {
00060         // Read a chunk from stdin
00061         if (std::fread(instance.GetBufferIn().data(), sizeof(float),
00062             sound4::bigvoice::InputSampleSize, stdin) != sound4::bigvoice::InputSampleSize) {
00063             std::cerr << "ProcessLoop: input stopped" << std::endl;
00064             stopthread=true;
00065             break;
00066         }
00067         if (stopthread) break;
00068         // Process it
00069         instance.ProcessAudio();
00070         // Write it to stdout
00071         if (fwrite(instance.GetBufferOut().data(), sizeof(float), sound4::bigvoice::OutputSampleSize,
00072             stdout) != sound4::bigvoice::OutputSampleSize) {
00073             std::cerr << "Failed to write audio output" << std::endl;
00074             stopthread=true;
00075         }
00076     }
00077 }
00078
00079 void Usage(const char *progname)
00080 {
00081     std::cerr << "Usage: " << progname << std::endl;;
00082     std::cerr << "  Reads audio from stdin and return processed audio in stdout" << std::endl;;
00083     std::cerr << "  Environment:" << std::endl;;
00084     std::cerr << "    - S4STORE_PATH          : path to store config [default: (no storage)]" <<
00085         std::endl;;
00086     std::cerr << "    - S4LOGINKEY           : SOUND4 Login key (or S4USERID)" << std::endl;;
00087     std::cerr << "    - RADIO_NAME          : SOUND4 Radio Name" << std::endl;;
00088     std::cerr << "    - S4_AWS_ACCESS_KEY_ID : SOUND4 Access Key ID" << std::endl;;
00089     std::cerr << "    - S4_AWS_SECRET_ACCESS_KEY: SOUND4 Access Key Secret" << std::endl;;
00090     std::cerr << std::endl;
00091     std::cerr << "Example:" << std::endl;
00092     std::cerr << "ffmpeg -re -i file.wav -ar " << sound4::bigvoice::GetSampleRate() << " -ac " <<
00093         sound4::bigvoice::GetChannelCount() << " -f f32le - | " << progname << " | ffplay -f f32le -ar " <<
00094         sound4::bigvoice::GetSampleRate() << " -ac " << sound4::bigvoice::GetChannelCount() << " -i -" << std::endl;;
00095     #if defined(_WIN32) || defined(WIN32)
```

```

00093         std::cerr << "ffmpeg -f dshow -i audio=\"Microphone\" -ar \"<sound4::bigvoice::GetSampleRate()<\"
-ac \"<sound4::bigvoice::GetChannelCount()<\" -f f32le - | \"<progrname>\" | ffplay -f f32le -ar
\"<sound4::bigvoice::GetSampleRate()<\" -ac \"<sound4::bigvoice::GetChannelCount()<\" -i -\" << std::endl;;
00094         #else
00095         std::cerr << "ffmpeg -f pulse -i default -ar \"<sound4::bigvoice::GetSampleRate()<\" -ac
\"<sound4::bigvoice::GetChannelCount()<\" -f f32le - | \"<progrname>\" | ffplay -f f32le -ar
\"<sound4::bigvoice::GetSampleRate()<\" -ac \"<sound4::bigvoice::GetChannelCount()<\" -i -\" << std::endl;;
00096         #endif
00097     }
00098
00099 void MyLogger(sound4::bigvoice::LogSeverity severity, const std::string& msg)
00100 {
00101     std::cerr << msg << std::endl;
00102 }
00103
00104 int main(int argc, const char **argv)
00105 {
00106     #if (BIGVOICE_HAS_WEBSERVER)
00107     int listen_port = 8080;
00108     #endif // (BIGVOICE_HAS_WEBSERVER)
00109     auto store_path_str=GetEnv("S4STORE_PATH");
00110
00111     // Get credentials from environment
00112     auto LoginKey=GetEnv("S4LOGINKEY");
00113     if (LoginKey.empty()) {
00114         LoginKey=GetEnv("S4USERID");
00115     }
00116     auto RadioName=GetEnv("RADIO_NAME");
00117     auto KeyId=GetEnv("S4_AWS_ACCESS_KEY_ID");
00118     auto AccessKey=GetEnv("S4_AWS_SECRET_ACCESS_KEY");
00119
00120     const char *progrname=argv[0];
00121
00122     if (argc>1) {
00123         if ((strcasecmp(argv[1], "-h")==0) || (strcasecmp(argv[1], "--help")==0)) {
00124             Usage(progrname);
00125             return 0;
00126         }
00127     }
00128     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00129         std::cerr << "Missing credentials in environment" << std::endl;
00130         Usage(progrname);
00131         return 1;
00132     }
00133     std::filesystem::path store_path;
00134     if (!store_path_str.empty()) {
00135         store_path = std::filesystem::path(store_path_str);
00136         if (!std::filesystem::exists(store_path) || !std::filesystem::is_directory(store_path)) {
00137             std::cerr << "Storage folder '\"<store_path.c_str()<\"' is not a valid directory" <<
std::endl;
00138             Usage(progrname);
00139             return 1;
00140         }
00141     }
00142
00143     sound4::bigvoice::SetLogSeverity(verbose5);
00144     sound4::bigvoice::SetLoggerCallback(MyLogger);
00145
00146     sound4::bigvoice::CInstance instance;
00147     instance.SetParam("ADMIN_USER", "admin");
00148     instance.SetParam("ADMIN_SECRET", "admin");
00149
00150     std::cerr << "Creating" << std::endl;
00151     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, store_path)) {
00152         std::cerr << "Failed to create the processing instance" << std::endl;
00153         return 1;
00154     }
00155
00156     #if (BIGVOICE_HAS_WEBSERVER)
00157     std::cerr << "Starting web server" << std::endl;
00158     if (!instance.StartWebServer(listen_port, 0)) {
00159         std::cerr << "Failed to create the web server on port " << listen_port << std::endl;
00160         return 1;
00161     }
00162     #endif // (BIGVOICE_HAS_WEBSERVER)
00163
00164     {
00165         auto client = instance.NewClient();
00166         auto answer = client->ProcessJson(R"JSON({\"get\":{\"processorname\":null}})JSON");
00167         std::cerr << "Request for processor name returned "<answer<<std::endl;
00168     }
00169
00170     ProcessLoop(instance);
00171
00172     std::cerr << "Stopping now" << std::endl;
00173     instance.Stop();
00174 }

```

```
00183     std::cerr << "End" << std::endl;
00184     return 0;
00185 }
```

11.5 sound4.bigvoice.cl-pipe-dyn.cpp File Reference

C++ sample using C++ interface wrapper with dynamic library loading.

```
#include <unistd.h>
#include <iostream>
#include <strings.h>
#include <sys/types.h>
#include <filesystem>
#include <fstream>
#include <sstream>
#include <io.h>
#include <fcntl.h>
#include "sound4.bigvoice.cl_dyn.hpp"
```

Classes

- class [MyPresetManager](#)
[Custom preset manager]

Functions

- std::string [GetEnv](#) (const std::string &name)
- void [ProcessLoop](#) ([sound4::bigvoice::dyn::CInstance](#) &instance)
- void [Usage](#) (const char *progname, [sound4::bigvoice::dyn::CDynLib](#) &dynlib)
- void [MyLogger](#) ([sound4::bigvoice::dyn::LogSeverity](#) severity, const std::string &msg)
- int [main](#) (int argc, const char **argv)
[Custom preset manager]

11.5.1 Function Documentation

11.5.1.1 GetEnv()

```
std::string GetEnv (
    const std::string & name )
```

Definition at line 51 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

11.5.1.2 main()

```
int main (
    int argc,
    const char ** argv )
```

[Dynamic loading]

[Dynamic loading]

[Setup and sanity checks]

[Setup and sanity checks]

[Custom preset manager loading]

[Custom preset manager loading]

[Create the processing instance]

[Create the processing instance]

[Communicating with the instance]

[Communicating with the instance]

Definition at line 194 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

11.5.1.3 MyLogger()

```
void MyLogger (
    sound4::bigvoice::dyn::LogSeverity severity,
    const std::string & msg )
```

Definition at line 113 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

11.5.1.4 ProcessLoop()

```
void ProcessLoop (
    sound4::bigvoice::dyn::CInstance & instance )
```

[Processing Loop]

[Processing Loop]

Definition at line 60 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

11.5.1.5 Usage()

```
void Usage (
    const char * progname,
    sound4::bigvoice::dyn::CDynLib & dynlib )
```

Definition at line 91 of file [sound4.bigvoice.cl-pipe-dyn.cpp](#).

11.6 sound4.bigvoice.cl-pipe-dyn.cpp

```
00001
00004 #include <unistd.h>
00005 #include <iostream>
00006 #include <strings.h>
00007 #include <sys/types.h>
00008 #include <filesystem>
00009 #include <fstream>
00010 #include <sstream>
00011 #if defined(_WIN32) || defined(WIN32)
00012     #include <io.h>
00013     #include <fcntl.h>
00014 #endif
00015
00016 #include "sound4.bigvoice.cl_dyn.hpp"
00017
00018 // -----
00051 std::string GetEnv(const std::string& name)
00052 {
00053     char *data=getenv(name.c_str());
00054     if (!data) return {};
00055     return std::string(data);
00056 }
00057
00058
00059 // Reads from stdin, write to stdout
00060 void ProcessLoop(sound4::bigvoice::dyn::CInstance &instance)
00061 {
00062     bool stopthread=false;
00063
00064     #if defined(_WIN32) || defined(WIN32)
00065         // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00066         _setmode(_fileno(stdin), _O_BINARY);
00067         _setmode(_fileno(stdout), _O_BINARY);
00068     #endif
00069
00071     while (!stopthread)
00072     {
00073         // Read a chunk from stdin
00074         if (std::freed(instance.GetBufferIn().data(), sizeof(float),
00075             sound4::bigvoice::dyn::InputSampleSize, stdin) != sound4::bigvoice::dyn::InputSampleSize) {
00076             std::cerr << "ProcessLoop: input stopped" << std::endl;
00077             stopthread=true;
00078             break;
00079         }
00080         // Process it
00081         instance.ProcessAudio();
00082         // Write it to stdout
00083         if (fwrite(instance.GetBufferOut().data(), sizeof(float),
00084             sound4::bigvoice::dyn::OutputSampleSize, stdout) != sound4::bigvoice::dyn::OutputSampleSize) {
00085             std::cerr << "Failed to write audio output" << std::endl;
00086             stopthread=true;
00087         }
00088     }
00089 }
00090
00091 void Usage(const char *progname, sound4::bigvoice::dyn::CDynLib &dynlib)
00092 {
00093     std::cerr << "Usage: " << progname << std::endl;;
00094     std::cerr << "  Reads audio from stdin and return processed audio in stdout" << std::endl;;
00095     std::cerr << "Environment:" << std::endl;;
00096     std::cerr << "  - S4STORE_PATH          : path to store config [default: (no storage)]" <<
00097         std::endl;;
00098     std::cerr << "  - S4LOGINKEY           : SOUND4 Login key (or S4USERID)" << std::endl;;
00099     std::cerr << "  - RADIO_NAME          : SOUND4 Radio Name" << std::endl;;
00100     std::cerr << "  - S4_AWS_ACCESS_KEY_ID : SOUND4 Access Key ID" << std::endl;;
00101     std::cerr << "  - S4_AWS_SECRET_ACCESS_KEY: SOUND4 Access Key Secret" << std::endl;;
00102     if (dynlib.IsOk()) {
00103         std::cerr << std::endl;
```

```

00103         std::cerr << "Example:"<< std::endl;
00104         std::cerr << "ffmpeg -re -i file.wav -ar "<<dynlib.GetSampleRate()<<" -ac
"<<dynlib.GetChannelCount()<<" -f f32le - | "<<progrname<<" | ffplay -f f32le -ar
"<<dynlib.GetSampleRate()<<" -ac "<<dynlib.GetChannelCount()<<" -i -" << std::endl;;
00105         #if defined(_WIN32) || defined(WIN32)
00106         std::cerr << "ffmpeg -f dshow -i audio=\"Microphone\" -ar "<<dynlib.GetSampleRate()<<" -ac
"<<dynlib.GetChannelCount()<<" -f f32le - | "<<progrname<<" | ffplay -f f32le -ar
"<<dynlib.GetSampleRate()<<" -ac "<<dynlib.GetChannelCount()<<" -i -" << std::endl;;
00107         #else
00108         std::cerr << "ffmpeg -f pulse -i default -ar "<<dynlib.GetSampleRate()<<" -ac
"<<dynlib.GetChannelCount()<<" -f f32le - | "<<progrname<<" | ffplay -f f32le -ar
"<<dynlib.GetSampleRate()<<" -ac "<<dynlib.GetChannelCount()<<" -i -" << std::endl;;
00109         #endif
00110     }
00111 }
00112
00113 void MyLogger(sound4::bigvoice::dyn::LogSeverity severity, const std::string& msg)
00114 {
00115     std::cerr << msg << std::endl;
00116 }
00117
00119 class MyPresetManager : public sound4::bigvoice::dyn::CPresetLoader {
00120 public:
00121     MyPresetManager() = default;
00122     virtual ~MyPresetManager() = default;
00123
00124     bool SetupPath(const std::filesystem::path& a_preset_path)
00125     {
00126         m_preset_path=a_preset_path;
00127         if ( !std::filesystem::exists(m_preset_path ) ) {
00128             return false;
00129         }
00130         std::error_code ec;
00131         if ( !std::filesystem::is_directory(m_preset_path, ec) ) {
00132             return false;
00133         }
00134         return true;
00135     }
00136     virtual bool IsReadOnly() override { return false; };
00137     virtual bool Exists(const std::filesystem::path &name) override
00138     {
00139         std::error_code ec;
00140         return std::filesystem::exists(GetRealPath(name), ec);
00141     }
00142     virtual bool Remove(const std::filesystem::path &name) override
00143     {
00144         std::error_code ec;
00145         return std::filesystem::remove(GetRealPath(name), ec);
00146     }
00147     virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) override
00148     {
00149         std::error_code ec;
00150         std::filesystem::rename(GetRealPath(from), GetRealPath(to), ec);
00151         return (!ec);
00152     }
00153     virtual std::vector<std::filesystem::path> GetAll() override
00154     {
00155         std::vector<std::filesystem::path> list;
00156         for (const auto & entry : std::filesystem::directory_iterator(m_preset_path.c_str(),
std::filesystem::directory_options::skip_permission_denied)) {
00157             if (entry.is_regular_file()) {
00158                 list.push_back(std::filesystem::relative(entry.path(), m_preset_path).native());
00159             }
00160         }
00161         return list;
00162     }
00163     virtual std::string Read(const std::filesystem::path &filename) override
00164     {
00165         std::ifstream f(GetRealPath(filename));
00166         if ( f.fail() ) {
00167             return std::string();
00168         }
00169         std::ostringstream ss;
00170         ss << f.rdbuf();
00171         return ss.str();
00172     }
00173     virtual bool Write(const std::filesystem::path &filename, const std::string &content) override
00174     {
00175         std::ofstream f(GetRealPath(filename));
00176         if ( f.fail() ) {
00177             return false;
00178         }
00179         f << content;
00180         if ( f.fail() ) {
00181             return false;
00182         }
00183         return true;

```

```

00184     }
00185 private:
00186     std::filesystem::path m_preset_path;
00187     std::filesystem::path GetRealPath(const std::filesystem::path &name)
00188     {
00189         return m_preset_path / name;
00190     }
00191 };
00192
00193
00194 int main(int argc, const char **argv)
00195 {
00196     #if (BIGVOICE_HAS_WEBSERVER)
00197         int listen_port = 8080;
00198     #endif // (BIGVOICE_HAS_WEBSERVER)
00199     auto store_path_str=GetEnv("S4STORE_PATH");
00200
00201     // Get credentials from environment
00202     auto LoginKey=GetEnv("S4LOGINKEY");
00203     if (LoginKey.empty()) {
00204         LoginKey=GetEnv("S4USERID");
00205     }
00206     auto RadioName=GetEnv("RADIO_NAME");
00207     auto KeyId=GetEnv("S4_AWS_ACCESS_KEY_ID");
00208     auto AccessKey=GetEnv("S4_AWS_SECRET_ACCESS_KEY");
00209
00210     const char *programe=argv[0];
00211
00212
00213     sound4::bigvoice::dyn::CDynLib dynlib;
00214     if (!dynlib.Load()) {
00215         std::cerr << "Failed to load processing library" << std::endl;
00216         return 1;
00217     }
00218
00219     if (argc>1) {
00220         if ((strcmp(argv[1],"-h")==0) || (strcmp(argv[1],"--help")==0)) {
00221             Usage(programe, dynlib);
00222             return 0;
00223         }
00224     }
00225
00226     if (LoginKey.empty() || RadioName.empty() || KeyId.empty() || AccessKey.empty()) {
00227         std::cerr << "Missing credentials in environment" << std::endl;
00228         Usage(programe, dynlib);
00229         return 1;
00230     }
00231
00232     std::filesystem::path store_path;
00233     if (!store_path_str.empty()) {
00234         store_path = std::filesystem::path(store_path_str);
00235         if (!std::filesystem::exists(store_path) || !std::filesystem::is_directory(store_path)) {
00236             std::cerr << "Storage folder '" << store_path.c_str() << "' is not a valid directory" <<
00237             std::endl;
00238             Usage(programe, dynlib);
00239             return 1;
00240         }
00241     }
00242
00243     sound4::bigvoice::dyn::SetLogSeverity(dynlib, verbose5);
00244     sound4::bigvoice::dyn::SetLoggerCallback(dynlib, MyLogger);
00245
00246     sound4::bigvoice::dyn::CInstance instance(dynlib);
00247     instance.SetParam("ADMIN_USER", "admin");
00248     instance.SetParam("ADMIN_SECRET", "admin");
00249
00250     MyPresetManager preset_manager;
00251     if (!store_path.empty()) {
00252         if (!preset_manager.SetupPath(store_path)) {
00253             std::cerr << "Failed to setup the custom preset manager on path " << store_path.c_str() <<
00254             std::endl;
00255             return 1;
00256         }
00257     }
00258     instance.SetPresetManager(&preset_manager);
00259
00260
00261     std::cerr << "Creating" << std::endl;
00262     if (!instance.Create(LoginKey, RadioName, KeyId, AccessKey, {})) {
00263         std::cerr << "Failed to create the processing instance" << std::endl;
00264         return 1;
00265     }
00266
00267
00268     #if (BIGVOICE_HAS_WEBSERVER)
00269     std::cerr << "Starting web server" << std::endl;
00270     if (!instance.StartWebServer(listen_port, 0)) {
00271         std::cerr << "Failed to create the web server on port " << listen_port << std::endl;
00272         return 1;
00273     }
00274     #endif // (BIGVOICE_HAS_WEBSERVER)
00275
00276
00277
00278
00279

```

```

00280     {
00282         auto client = instance.NewClient();
00283         auto answer = client->ProcessJson(R"JSON({"get":{"processorname":null}})JSON");
00284         std::cerr << "Request for processor name returned " << answer << std::endl;
00286     }
00287
00288     ProcessLoop(instance);
00289
00290     std::cerr << "Stopping now" << std::endl;
00291     instance.Stop();
00292
00293     std::cerr << "End" << std::endl;
00294     return 0;
00295 }

```

11.7 sound4.bigvoice.cl-pipe.cpp File Reference

C++ sample using only C interface.

```

#include <unistd.h>
#include <iostream>
#include <strings.h>
#include <sys/types.h>
#include <thread>
#include <experimental/filesystem>
#include <io.h>
#include <fcntl.h>
#include <stringapiset.h>
#include <sound4.bigvoice.cl.h>

```

Namespaces

- [std](#)

Macros

- #define [SAMPLE_RATE](#) 48000
- #define [CHUNK_SIZE](#) 64
- #define [CHANNEL_COUNT](#) 2

Functions

- void [ProcessLoop](#) (struct [bigvoice_CInstance](#) *pInstance)
- void [Usage](#) (const char *progname)
- void [MyLogger](#) ([bigvoice_LogSeverity](#) severe[[maybe_unused]], const char *str)
- int [main](#) (int argc, char *const argv[])

11.7.1 Macro Definition Documentation

11.7.1.1 CHANNEL_COUNT

```
#define CHANNEL_COUNT 2
```

Definition at line 28 of file [sound4.bigvoice.cl-pipe.cpp](#).

11.7.1.2 CHUNK_SIZE

```
#define CHUNK_SIZE 64
```

Definition at line 27 of file [sound4.bigvoice.cl-pipe.cpp](#).

11.7.1.3 SAMPLE_RATE

```
#define SAMPLE_RATE 48000
```

Definition at line 26 of file [sound4.bigvoice.cl-pipe.cpp](#).

11.7.2 Function Documentation

11.7.2.1 main()

```
int main (  
    int argc,  
    char *const argv[] )
```

[Setup and sanity checks]

[Setup and sanity checks]

[Create the processing instance]

[Create the processing instance]

[Stopping the processing instance]

[Stopping the processing instance]

Definition at line 90 of file [sound4.bigvoice.cl-pipe.cpp](#).

11.7.2.2 MyLogger()

```
void MyLogger (
    bigvoice_LogSeverity severe] [[maybe_unused],
    const char * str )
```

Definition at line 84 of file [sound4.bigvoice.cl-pipe.cpp](#).

11.7.2.3 ProcessLoop()

```
void ProcessLoop (
    struct bigvoice_CInstance * pInstance )
```

[Processing Loop]

[Processing Loop]

Definition at line 31 of file [sound4.bigvoice.cl-pipe.cpp](#).

11.7.2.4 Usage()

```
void Usage (
    const char * progname )
```

Definition at line 64 of file [sound4.bigvoice.cl-pipe.cpp](#).

11.8 sound4.bigvoice.cl-pipe.cpp

```
00001
00004 #include <unistd.h>
00005 #include <iostream>
00006 #include <strings.h>
00007 #include <sys/types.h>
00008 #include <thread>           // NOTE: needs pthread link
00009 // Choose good filesystem version depending on compiler.
00010 #if __has_include(<filesystem>)
00011     #include <filesystem>
00012 #else //if __cpp_lib_experimental_filesystem
00013     // We need the alias from std::experimental::filesystem to std::filesystem
00014     #include <experimental/filesystem>
00015     namespace std {
00016         namespace filesystem = experimental::filesystem;
00017     }
00018 #endif
00019 #if defined(_WIN32) || defined(WIN32)
00020     #include <io.h>
00021     #include <fcntl.h>
00022     #include <stringapiset.h> // UTF-8 conversion
00023 #endif
00024 #include <sound4.bigvoice.cl.h>
00025
00026 #define SAMPLE_RATE      48000
00027 #define CHUNK_SIZE       64
00028 #define CHANNEL_COUNT    2
00029
00030 // Reads from stdin, write to stdout
00031 void ProcessLoop(struct bigvoice_CInstance *pInstance)
00032 {
00034     bool stopthread=false;
```

```

00035 alignas(64) float audio_in[CHUNK_SIZE*CHANNEL_COUNT];
00036 alignas(64) float audio_out[CHUNK_SIZE*CHANNEL_COUNT];
00037
00038 #if defined(_WIN32) || defined(WIN32)
00039     // Windows: The stdin, stdout, and stderr streams always open in text mode by default
00040     _setmode(_fileno(stdin), _O_BINARY);
00041     _setmode(_fileno(stdout), _O_BINARY);
00042 #endif
00043
00044 while (!stopthread)
00045 {
00046     // Read a chunk from stdin
00047     if (std::fread(&audio_in[0], sizeof(float), CHUNK_SIZE*CHANNEL_COUNT, stdin) !=
CHUNK_SIZE*CHANNEL_COUNT) {
00048         std::cerr << "ProcessLoop: input stopped" << std::endl;
00049         stopthread=true;
00050         break;
00051     }
00052     if (stopthread) break;
00053     // Process it
00054     bigvoice_ProcessAudio(pInstance, audio_in, audio_out);
00055     // Write it to stdout
00056     if (fwrite(&audio_out[0], sizeof(float), CHUNK_SIZE*CHANNEL_COUNT, stdout) !=
CHUNK_SIZE*CHANNEL_COUNT) {
00057         std::cerr << "Failed to write audio output" << std::endl;
00058         stopthread=true;
00059     }
00060 }
00061 }
00062 }
00063
00064 void Usage(const char *programe)
00065 {
00066     std::cerr << "Usage: " << programe << std::endl;;
00067     std::cerr << "  Reads audio from stdin and return processed audio in stdout" << std::endl;
00068     std::cerr << "Environment:" << std::endl;;
00069     std::cerr << "  - S4STORE_PATH           : path to store config [default: ./state]" << std::endl;;
00070     std::cerr << "  - S4LOGINKEY           : SOUND4 Login key (or S4USERID)" << std::endl;;
00071     std::cerr << "  - RADIO_NAME          : SOUND4 Radio Name" << std::endl;;
00072     std::cerr << "  - S4_AWS_ACCESS_KEY_ID : SOUND4 Access Key ID" << std::endl;;
00073     std::cerr << "  - S4_AWS_SECRET_ACCESS_KEY: SOUND4 Access Key Secret" << std::endl;;
00074     std::cerr << std::endl;
00075     std::cerr << "Example:" << std::endl;
00076     std::cerr << "ffmpeg -re -i file.wav -ar "<<SAMPLE_RATE<<" -ac "<<CHANNEL_COUNT<<" -f f32le - |
"<<programe<<" | ffplay -f f32le -ar "<<SAMPLE_RATE<<" -ac "<<CHANNEL_COUNT<<" -i -" << std::endl;;
00077     #if defined(_WIN32) || defined(WIN32)
00078     std::cerr << "ffmpeg -f dshow -i audio=\\\"Microphone\\\" -ar "<<SAMPLE_RATE<<" -ac "<<CHANNEL_COUNT<<"
-f f32le - | "<<programe<<" | ffplay -f f32le -ar "<<SAMPLE_RATE<<" -ac "<<CHANNEL_COUNT<<" -i -" <<
std::endl;;
00079     #else
00080     std::cerr << "ffmpeg -f pulse -i default -ar "<<SAMPLE_RATE<<" -ac "<<CHANNEL_COUNT<<" -f f32le - |
"<<programe<<" | ffplay -f f32le -ar "<<SAMPLE_RATE<<" -ac "<<CHANNEL_COUNT<<" -i -" << std::endl;;
00081     #endif
00082 }
00083
00084 void MyLogger(bigvoice_LogSeverity severe [[maybe_unused]], const char *str)
00085 {
00086     std::cerr << str << std::endl;
00087 }
00088
00089
00090 int main(int argc, char *const argv[])
00091 {
00092     struct bigvoice_CInstance *pInstance = NULL;
00093     const char *store_path_str=getenv("S4STORE_PATH");
00094     // Get credentials from environment
00095     char *pLoginKey=getenv("S4LOGINKEY");
00096     if (!pLoginKey) {
00097         pLoginKey=getenv("S4USERID");
00098     }
00099     char *pRadioName=getenv("RADIO_NAME");
00100     char *pKeyId=getenv("S4_AWS_ACCESS_KEY_ID");
00101     char *pAccessKey=getenv("S4_AWS_SECRET_ACCESS_KEY");
00102
00103     const char *programe=argv[0];
00104
00105     if (argc>1) {
00106         if ((strcmp(argv[1], "-h")==0) || (strcmp(argv[1], "--help")==0)) {
00107             Usage(programe);
00108             return 0;
00109         }
00110     }
00111     if (!pLoginKey || !pRadioName || !pKeyId || !pAccessKey) {
00112         std::cerr << "Missing credentials in environment" << std::endl;
00113         Usage(programe);
00114         return 1;
00115     }
00116     std::filesystem::path fs_store_path;

```

```

00117     if (!store_path_str || store_path_str[0]==0) {
00118         fs_store_path=std::filesystem::current_path() / "state";
00119     } else {
00120         fs_store_path = std::filesystem::path(store_path_str);
00121     }
00122     if (!std::filesystem::exists(fs_store_path) || !std::filesystem::is_directory(fs_store_path)) {
00123         std::cerr << "Storage folder '"<fs_store_path.c_str()<"' is not a valid directory" << std::endl;
00124         Usage(progname);
00125         return 1;
00126     }
00127     std::string store_path;
00128     #ifdef _WIN32
00129     {
00130         // Convert path to UTF-8
00131         std::wstring wstr(fs_store_path.wstring());
00132         if (wstr.empty()) return {};
00133         int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL, NULL);
00134         std::string str(size, 0);
00135         WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00136         store_path=str;
00137     }
00138     #else // !_WIN32
00139         store_path=fs_store_path;
00140     #endif // !_WIN32
00141
00142     // Optional: Use our own logging, instead of default color output to stderr.
00143     //bigvoice_SetLoggerCallback(&MyLogger);
00144
00145     //bigvoice_SetLogSeverity(bigvoice_LogSeverity::info);
00146     bigvoice_SetLogSeverity(bigvoice_LogSeverity::verbose5);
00147
00148     std::cerr << "Process version: " << bigvoice_GetVersion() << std::endl;
00149     // Do some sanity checks
00150     if (bigvoice_GetSampleRate() != SAMPLE_RATE) {
00151         std::cerr << "Bad library sampling rate, expecting '"<SAMPLE_RATE<"' but has "<
bigvoice_GetSampleRate() << std::endl;
00152         return 1;
00153     }
00154     if (bigvoice_GetChunkSizeInFrames() != CHUNK_SIZE) {
00155         std::cerr << "Bad library frame size, expecting '"<CHUNK_SIZE<"' but has "<
bigvoice_GetChunkSizeInFrames() << std::endl;
00156         return 1;
00157     }
00158     if (bigvoice_GetChannelCount() != CHANNEL_COUNT) {
00159         std::cerr << "Bad library channel count, expecting '"<CHANNEL_COUNT<"' but has "<
bigvoice_GetChannelCount() << std::endl;
00160         return 1;
00161     }
00162
00163     // Setup parameters
00164     struct bigvoice_CParameters *params = bigvoice_NewParameters();
00165     // Optional: activate web authentication, set admin password
00166     // This creates the authentication only for a new storage path
00167     bigvoice_SetParameter(params, "ADMIN_USER", "admin");
00168     bigvoice_SetParameter(params, "ADMIN_SECRET", "admin");
00169
00170     // Create the processing instance
00171     pInstance=bigvoice_InitProcess2(pLoginKey, pRadioName, pKeyId, pAccessKey, store_path.c_str(),
params);
00172     bigvoice_FreeParameters(params);
00173     if (!pInstance) {
00174         std::cerr << "Failed to create the process instance"< std::endl;
00175         return 1;
00176     }
00177
00178     // Start the update thread, without JSON server (low priority)
00179     std::thread updater([pInstance]() {
00180         // will returned when stopped or destroyed
00181         std::cerr << "bigvoice_StartUpdateThread (no json port)" << std::endl;
00182         bigvoice_StartUpdateThread(pInstance, 0);
00183         std::cerr << "bigvoice_StartUpdateThread END" << std::endl;
00184     });
00185
00186     // This is not mandatory, but better be sure we are ready right now
00187     std::cerr << "Waiting update thread" << std::endl;
00188     if (bigvoice_WaitUpdateThreadReady(pInstance, 5000)<0) {
00189         std::cerr << "Timeout to create the process instance"< std::endl;
00190     }
00191     std::cerr << "Waiting update thread DONE" << std::endl;
00192
00193     #if (BIGVOICE_HAS_WEBSERVER)
00194     // Starts a web server, listening on port http://localhost:8080, https://localhost:8443
00195     std::cerr << "Starting web server" << std::endl;
00196     int listenport= 8080;
00197     int listenport_secure= 8443;
00198     uint64_t webserver= SOUND4_INVALID_WEBSERVER_ID;
00199     if (listenport || listenport_secure) {

```

```

00204     webserver = bigvoice_Webserver(listenport,listenport_secure,pInstance);
00205     if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
00206         std::cerr << "Starting web server FAILED" << std::endl;
00207     } else {
00208         int web_status=bigvoice_Webserver_Status(webserver);
00209         if (listenport) {
00210             if (web_status & SOUND4_WEBSERVER_HTTP_OK) {
00211                 std::cerr << "Web server listening on port " << listenport;
00212             } else {
00213                 std::cerr << "Web server failed to listen on port " << listenport;
00214             }
00215         }
00216         if (listenport_secure) {
00217             if (web_status & SOUND4_WEBSERVER_HTTPS_OK) {
00218                 std::cerr << "Web server listening on secure port " << listenport_secure;
00219             } else {
00220                 std::cerr << "Web server failed to listen on secure port " << listenport_secure;
00221             }
00222         }
00223         std::cerr << "Starting web server DONE" << std::endl;
00224     }
00225 }
00227 #endif // (BIGVOICE_HAS_WEBSERVER)
00228
00229 // Force load a factory preset now
00230 // Could be used instead of setting a store_path
00231 /*{
00232     bigvoice_CClientInstance *pClient=bigvoice_NewClient(pInstance);
00233     const char *answer=bigvoice_ProcessJson(pClient, "{ \"factoryload\": \"All Programs\\\" }",
00234     NULL);
00235     std::cerr << "Loaded factory preset answer: " << answer<< std::endl;
00236     bigvoice_FreeJsonAnswer(answer);
00237     bigvoice_DeleteClient(pClient);
00238 }*/
00239
00240 std::cerr << "Estimated delay " << ( bigvoice_GetEstimatedDelay(pInstance)*1000.0/SAMPLE_RATE ) << "
ms" << std::endl;
00241
00242     ProcessLoop(pInstance);
00243
00245 #if (BIGVOICE_HAS_WEBSERVER)
00246     // Stopping the webserver
00247     std::cerr << "Stopping web server" << std::endl;
00248     bigvoice_Webserver_Stop(webserver, 1000);
00249     std::cerr << "Stopping web server DONE" << std::endl;
00250 #endif // (BIGVOICE_HAS_WEBSERVER)
00251
00252     // Stopping the processing instance
00253     std::cerr << "Stopping Update thread" << std::endl;
00254     bigvoice_StopUpdateThread(pInstance); // this will exit the update thread
00255     std::cerr << "Stopping Update thread DONE" << std::endl;
00256
00257     updater.join();
00258
00259     bigvoice_ExitProcess(pInstance);
00260
00261     return 0;
00262 }
00263 }

```

11.9 sound4.bigvoice.cl.h File Reference

C interface for library.

```
#include <stdint.h>
```

Macros

- #define `BIGVOICE_DECL __declspec(dllimport)`
- #define `BIGVOICE_HAS_WEBSERVER` 1
- #define `BIGVOICE_HAS_CLOUDBUS` 1
- #define `BIGVOICE_AUDIOFRAME_COUNT` 12
- #define `BIGVOICE_CHANNEL_COUNT` 1

- `#define BIGVOICE_AUDIO_INPUT_COUNT 1`
- `#define BIGVOICE_AUDIO_OUTPUT_COUNT 1`
- `#define BIGVOICE_SAMPLE_RATE 48000`
- `#define SOUND4_INVALID_WEBSERVER_ID ((uint64_t)-1)`
- `#define SOUND4_WEBSERVER_HTTP_OK (1<<0)`
- `#define SOUND4_WEBSERVER_HTTPS_OK (1<<1)`

Typedefs

- `typedef void(* bigvoice_loggerfn) (enum bigvoice_LogSeverity, const char *)`
- `typedef wchar_t fs_char`
- `typedef char *(* bigvoice_storage_reader) (const fs_char *filename, void *userdata)`
- `typedef void(* bigvoice_storage_reader_free) (char *content, void *userdata)`
- `typedef int(* bigvoice_storage_writer) (const fs_char *filename, const char *content, void *userdata)`
- `typedef int(* bigvoice_storage_exists) (const fs_char *filename, void *userdata)`
- `typedef fs_char *(* bigvoice_storage_getall) (void *userdata)`
- `typedef void(* bigvoice_storage_getall_free) (fs_char **all, void *userdata)`
- `typedef int(* bigvoice_storage_remove) (const fs_char *filename, void *userdata)`
- `typedef int(* bigvoice_storage_rename) (const fs_char *from, const fs_char *to, void *userdata)`

Enumerations

- `enum bigvoice_LogSeverity {
none = 0 , fatal = 1 , error = 2 , warning = 3 ,
info = 4 , verbose = 5 , verbose2 = 6 , verbose3 = 7 ,
verbose4 = 8 , verbose5 = 9 }`
- `enum bigvoice_PresetChange_Kind { change_kind_created = 1 , change_kind_modified , change_kind_deleted }`
- `enum bigvoice_SampleFormat {
INVALID_FORMAT =0 , S16_LE , S16_BE , S24_LE ,
S24_BE , S32_LE , S32_BE , F32_LE ,
F32_BE , S16_NATIVE =S16_LE , S24_NATIVE =S24_LE , S32_NATIVE =S32_LE ,
F32_NATIVE =F32_LE }`

Functions

- `const char * bigvoice_GetVersion ()`
- `unsigned int bigvoice_GetChunkSizeInFrames ()`
- `unsigned int * bigvoice_GetPossibleChunkSizeInFrames ()`
- `unsigned int bigvoice_GetChannelCount ()`
- `unsigned int bigvoice_GetAudioInputCount ()`
- `unsigned int bigvoice_GetAudioOutputCount ()`
- `unsigned int bigvoice_GetSampleRate ()`
- `void bigvoice_SetLoggerCallback (bigvoice_loggerfn logger)`
- `void bigvoice_SetLogSeverity (enum bigvoice_LogSeverity severity)`
- `struct bigvoice_CParameters * bigvoice_NewParameters ()`
- `void bigvoice_FreeParameters (struct bigvoice_CParameters *params)`
- `void bigvoice_SetParameter (struct bigvoice_CParameters *params, const char *name, const char *value)`
- `const char * bigvoice_GetParameter (struct bigvoice_CParameters *params, const char *name)`
- `void bigvoice_FreeParameterValue (const char *value)`
- `struct bigvoice_CInstance * bigvoice_InitProcess (const char *LoginKey, const char *RadioName, const char *AWS_Access_ID, const char *AWS_Access_Secret, const char *save_path)`

- struct [bigvoice_CInstance](#) * [bigvoice_InitProcess2](#) (const char *LoginKey, const char *RadioName, const char *AWS_Access_ID, const char *AWS_Access_Secret, const char *save_path, const struct [bigvoice_CParameters](#) *parameters)
- struct [bigvoice_CInstance](#) * [bigvoice_InitProcess3](#) (const char *LoginKey, const char *RadioName, const char *AWS_Access_ID, const char *AWS_Access_Secret, const char *save_path, const struct [bigvoice_CParameters](#) *parameters, unsigned int frames_per_chunk)
- unsigned int [bigvoice_GetProcessChunkFrames](#) (struct [bigvoice_CInstance](#) *instance)
- int [bigvoice_TerminateProcess](#) (struct [bigvoice_CInstance](#) *instance)
- void [bigvoice_ExitProcess](#) (struct [bigvoice_CInstance](#) *instance)
- struct [bigvoice_CBus](#) * [bigvoice_NewBus](#) ()
- void [bigvoice_FreeBus](#) (struct [bigvoice_CBus](#) *bus)
- void [bigvoice_SetInstanceBus](#) (struct [bigvoice_CParameters](#) *params, struct [bigvoice_CBus](#) *bus)
- void [bigvoice_SetPresetManager](#) (struct [bigvoice_CParameters](#) *params, [bigvoice_storage_reader](#) reader, [bigvoice_storage_reader_free](#), [bigvoice_storage_writer](#) writer, [bigvoice_storage_exists](#) exists, [bigvoice_storage_getall](#) getall, [bigvoice_storage_getall_free](#) getall_free, [bigvoice_storage_remove](#) remove, [bigvoice_storage_rename](#) rename, int IsReadOnly, void *userdata)
- void [bigvoice_PresetManager_InformChange](#) (struct [bigvoice_CInstance](#) *instance, const fs_char *relative↵_path, enum [bigvoice_PresetChange_Kind](#) change_kind)
- void [bigvoice_SetMetadata](#) (struct [bigvoice_CInstance](#) *instance, const char *key, const char *value)
- const char ** [bigvoice_GetMetadataInfos](#) (struct [bigvoice_CInstance](#) *instance)
- void [bigvoice_FreeMetadataInfos](#) (struct [bigvoice_CInstance](#) *instance, const char **infos)
- void [bigvoice_StartUpdateThread](#) (struct [bigvoice_CInstance](#) *instance, unsigned int port)
- void [bigvoice_StopUpdateThread](#) (struct [bigvoice_CInstance](#) *instance)
- int [bigvoice_WaitUpdateThreadReady](#) (struct [bigvoice_CInstance](#) *instance, int milliseconds)
- void [bigvoice_ProcessAudio](#) (struct [bigvoice_CInstance](#) *instance, const float *input, float *output)
- void [bigvoice_ProcessAudio_Planar](#) (struct [bigvoice_CInstance](#) *instance, float const *const *input, float *const *output)
- float * [bigvoice_GetBufferIn](#) (struct [bigvoice_CInstance](#) *instance)
- float * [bigvoice_GetBufferOut](#) (struct [bigvoice_CInstance](#) *instance)
- unsigned int [bigvoice_GetEstimatedDelay](#) (struct [bigvoice_CInstance](#) *instance)
- const char * [bigvoice_GetFormatName](#) (const enum [bigvoice_SampleFormat](#) fmt)
- enum [bigvoice_SampleFormat](#) [bigvoice_GetFormatFromName](#) (const char *name)
- unsigned int [bigvoice_GetBytesFromFormat](#) (const enum [bigvoice_SampleFormat](#) fmt)
- unsigned int [bigvoice_GetMaxPacketFrame](#) (struct [bigvoice_CInstance](#) *instance)
- unsigned int [bigvoice_AddAudio](#) (struct [bigvoice_CInstance](#) *instance, const uint8_t *payload, unsigned int nFrame, enum [bigvoice_SampleFormat](#) fmt)
- unsigned int [bigvoice_AddPadAudio](#) (struct [bigvoice_CInstance](#) *instance)
- unsigned int [bigvoice_GetOutputCount](#) (struct [bigvoice_CInstance](#) *instance)
- unsigned int [bigvoice_GetAudio](#) (struct [bigvoice_CInstance](#) *instance, uint8_t *payload, unsigned int max↵_nFrame, enum [bigvoice_SampleFormat](#) fmt)
- void [bigvoice_AudioConvertFrom](#) (const uint8_t *payload, float *output, size_t nSpl, enum [bigvoice_SampleFormat](#) fmt)
- void [bigvoice_AudioConvertTo](#) (const float *input, uint8_t *payload, size_t nSpl, enum [bigvoice_SampleFormat](#) fmt)
- void [bigvoice_StereoToMono](#) (const float *input, float *output, size_t nFrame)
- void [bigvoice_MonoToStereo](#) (const float *input, float *output, size_t nFrame)
- void [bigvoice_StereoToMono_Planar](#) (const float *inputL, const float *inputR, float *output, size_t nFrame)
- void [bigvoice_MonoToStereo_Planar](#) (const float *input, float *outputL, float *outputR, size_t nFrame)
- void [bigvoice_AudioMonoFromLiveStereo](#) (const uint8_t *payload, float *output)
- void [bigvoice_AudioMonoToLiveStereo](#) (const float *input, uint8_t *payload)
- struct [bigvoice_CClientInstance](#) * [bigvoice_NewClient](#) (struct [bigvoice_CInstance](#) *instance)
- void [bigvoice_DeleteClient](#) (struct [bigvoice_CClientInstance](#) *client)
- const char * [bigvoice_ProcessJson](#) (struct [bigvoice_CClientInstance](#) *client, const char *json_str, int *need_save)
- void [bigvoice_FreeJsonAnswer](#) (const char *json_str)

- int [bigvoice_SaveState](#) (struct [bigvoice_CInstance](#) *instance)
- uint64_t [bigvoice_Webserver_tcp](#) (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port)
- uint64_t [bigvoice_Webserver_tcp2](#) (unsigned int listenport, unsigned int listenport_secure, const char *socket_ip, unsigned int socket_port, const struct [bigvoice_CParameters](#) *parameters)
- uint64_t [bigvoice_Webserver](#) (unsigned int listenport, unsigned int listenport_secure, struct [bigvoice_CInstance](#) *instance)
- int [bigvoice_Webserver_Stop](#) (uint64_t id, int timeout_ms)
- int [bigvoice_Webserver_Status](#) (uint64_t id)

11.9.1 Macro Definition Documentation

11.9.1.1 BIGVOICE_DECL

```
#define BIGVOICE_DECL __declspec(dllimport)
```

Definition at line 14 of file [sound4.bigvoice.cl.h](#).

11.9.1.2 BIGVOICE_HAS_CLOUDBUS

```
#define BIGVOICE_HAS_CLOUDBUS 1
```

Definition at line 19 of file [sound4.bigvoice.cl.h](#).

11.9.1.3 BIGVOICE_HAS_WEBSERVER

```
#define BIGVOICE_HAS_WEBSERVER 1
```

Definition at line 18 of file [sound4.bigvoice.cl.h](#).

11.10 sound4.bigvoice.cl.h

```

00001
00004 #pragma once
00005
00006 #include <stdint.h>
00007 #ifdef __unix__
00008     #include <sys/types.h>
00009     #define BIGVOICE_DECL __attribute__((visibility("default")))
00010 #elif defined(_WIN32) || defined(WIN32)
00011     #ifdef BUILDING_BIGVOICE_DLL
00012         #define BIGVOICE_DECL __declspec(dllexport)
00013     #else
00014         #define BIGVOICE_DECL __declspec(dllimport)
00015     #endif
00016 #endif
00017
00018 #define BIGVOICE_HAS_WEBSERVER 1
00019 #define BIGVOICE_HAS_CLOUDBUS 1
00020
00021 #ifdef __cplusplus
00022 extern "C" {
00023 #endif // __cplusplus
00024
00068 // -----
00087 // -----
00140 // -----
00141
00142 // -----
00154 #define BIGVOICE_AUDIOFRAME_COUNT 12
00160 #define BIGVOICE_CHANNEL_COUNT 1
00166 #define BIGVOICE_AUDIO_INPUT_COUNT 1
00172 #define BIGVOICE_AUDIO_OUTPUT_COUNT 1
00178 #define BIGVOICE_SAMPLE_RATE 48000
00179
00185 BIGVOICE_DECL const char *bigvoice_GetVersion();
00186
00199 BIGVOICE_DECL unsigned int bigvoice_GetChunkSizeInFrames();
00200
00209 BIGVOICE_DECL unsigned int* bigvoice_GetPossibleChunkSizeInFrames();
00210
00218 BIGVOICE_DECL unsigned int bigvoice_GetChannelCount();
00219
00228 BIGVOICE_DECL unsigned int bigvoice_GetAudioInputCount();
00229
00238 BIGVOICE_DECL unsigned int bigvoice_GetAudioOutputCount();
00239
00247 BIGVOICE_DECL unsigned int bigvoice_GetSampleRate();
00248
00252 enum bigvoice_LogSeverity {
00253     none = 0,
00254     fatal = 1,
00255     error = 2,
00256     warning = 3,
00257     info = 4,
00258     verbose = 5,
00259     verbose2 = 6,
00260     verbose3 = 7,
00261     verbose4 = 8,
00262     verbose5 = 9,
00263 };
00264
00271 typedef void (*bigvoice_loggerfn) (enum bigvoice_LogSeverity, const char *);
00272
00282 BIGVOICE_DECL void bigvoice_SetLoggerCallback( bigvoice_loggerfn logger );
00283
00291 BIGVOICE_DECL void bigvoice_SetLogSeverity( enum bigvoice_LogSeverity severity );
00292
00295 // -----
00301 // -----
00305 struct bigvoice_CInstance;
00306
00310 struct bigvoice_CParameters;
00311
00319 BIGVOICE_DECL struct bigvoice_CParameters *bigvoice_NewParameters();
00320
00324 BIGVOICE_DECL void bigvoice_FreeParameters(struct bigvoice_CParameters *params);
00325
00337 BIGVOICE_DECL void bigvoice_SetParameter(struct bigvoice_CParameters *params, const char *name, const
char *value);
00338
00344 BIGVOICE_DECL const char *bigvoice_GetParameter(struct bigvoice_CParameters *params, const char
*name);
00345
00351 BIGVOICE_DECL void bigvoice_FreeParameterValue(const char *value);
00352
00360 BIGVOICE_DECL struct bigvoice_CInstance *bigvoice_InitProcess(

```

```

00361         const char *LoginKey,
00362         const char *RadioName,
00363         const char *AWS_Access_ID,
00364         const char *AWS_Access_Secret,
00365         const char *save_path
00366     );
00367
00375 BIGVOICE_DECL struct bigvoice_CInstance *bigvoice_InitProcess2(
00376     const char *LoginKey,
00377     const char *RadioName,
00378     const char *AWS_Access_ID,
00379     const char *AWS_Access_Secret,
00380     const char *save_path,
00381     const struct bigvoice_CParameters *parameters
00382 );
00383
00428 BIGVOICE_DECL struct bigvoice_CInstance *bigvoice_InitProcess3(
00429     const char *LoginKey,
00430     const char *RadioName,
00431     const char *AWS_Access_ID,
00432     const char *AWS_Access_Secret,
00433     const char *save_path,
00434     const struct bigvoice_CParameters *parameters,
00435     unsigned int frames_per_chunk
00436 );
00437
00449 BIGVOICE_DECL unsigned int bigvoice_GetProcessChunkFrames(struct bigvoice_CInstance *instance);
00450
00460 BIGVOICE_DECL int bigvoice_TerminateProcess(struct bigvoice_CInstance *instance);
00461
00467 BIGVOICE_DECL void bigvoice_ExitProcess(struct bigvoice_CInstance *instance);
00468
00471 #if BIGVOICE_HAS_CLOUDBUS
00472 // -----
00481 struct bigvoice_CBus;
00482
00488 BIGVOICE_DECL struct bigvoice_CBus* bigvoice_NewBus();
00489
00495 BIGVOICE_DECL void bigvoice_FreeBus(struct bigvoice_CBus *bus);
00496
00506 BIGVOICE_DECL void bigvoice_SetInstanceBus(struct bigvoice_CParameters *params, struct bigvoice_CBus
    *bus);
00507
00508
00510 #endif // BIGVOICE_HAS_CLOUDBUS
00511
00512 // -----
00535 // -----
00536
00537 // OS dependent filesystem path char type
00538 #ifdef _WIN32
00539     typedef wchar_t fs_char;
00540 #else // !_WIN32
00541     typedef char fs_char;
00542 #endif // !_WIN32
00543
00551 typedef char * (*bigvoice_storage_reader) (const fs_char *filename, void* userdata);
00552
00559 typedef void (*bigvoice_storage_reader_free) (char *content, void* userdata);
00560
00569 typedef int (*bigvoice_storage_writer) (const fs_char *filename, const char *content, void* userdata);
00570
00578 typedef int (*bigvoice_storage_exists) (const fs_char *filename, void* userdata);
00579
00587 typedef fs_char** (*bigvoice_storage_getall) (void* userdata);
00588
00595 typedef void (*bigvoice_storage_getall_free) (fs_char**all, void* userdata);
00596
00604 typedef int (*bigvoice_storage_remove) (const fs_char *filename, void* userdata);
00605
00614 typedef int (*bigvoice_storage_rename) (const fs_char *from, const fs_char *to, void* userdata);
00615
00633 BIGVOICE_DECL void bigvoice_SetPresetManager(struct bigvoice_CParameters *params
00634     , bigvoice_storage_reader reader, bigvoice_storage_reader_free
00635     , bigvoice_storage_writer writer
00636     , bigvoice_storage_exists exists
00637     , bigvoice_storage_getall getall, bigvoice_storage_getall_free getall_free
00638     , bigvoice_storage_remove remove
00639     , bigvoice_storage_rename rename
00640     , int IsReadOnly
00641     , void* userdata
00642 );
00643
00647 enum bigvoice_PresetChange_Kind {
00648     change_kind_created = 1,
00649     change_kind_modified ,
00650     change_kind_deleted ,

```

```

00651 };
00652
00661 BIGVOICE_DECL void bigvoice_PresetManager_InformChange(struct bigvoice_CInstance *instance, const
fs_char *relative_path, enum bigvoice_PresetChange_Kind change_kind);
00662
00665 // -----
00673 // -----
00674
00688 BIGVOICE_DECL void bigvoice_SetMetadata(struct bigvoice_CInstance *instance, const char* key, const
char* value);
00689
00697 BIGVOICE_DECL const char** bigvoice_GetMetadataInfos(struct bigvoice_CInstance *instance);
00698
00705 BIGVOICE_DECL void bigvoice_FreeMetadataInfos(struct bigvoice_CInstance *instance, const char**
infos);
00706
00709 // -----
00714 // -----
00715
00728 BIGVOICE_DECL void bigvoice_StartUpdateThread(struct bigvoice_CInstance *instance, unsigned int port);
00729
00735 BIGVOICE_DECL void bigvoice_StopUpdateThread(struct bigvoice_CInstance *instance);
00736
00744 BIGVOICE_DECL int bigvoice_WaitUpdateThreadReady(struct bigvoice_CInstance *instance, int
milliseconds);
00745
00748 // -----
00753 // -----
00775 BIGVOICE_DECL void bigvoice_ProcessAudio(struct bigvoice_CInstance *instance, const float *input,
float *output );
00776
00795 BIGVOICE_DECL void bigvoice_ProcessAudio_Planar(struct bigvoice_CInstance *instance
, float const * const *input
, float * const *output);
00796
00797 , float * const *output);
00798
00811 BIGVOICE_DECL float *bigvoice_GetBufferIn(struct bigvoice_CInstance *instance);
00812
00825 BIGVOICE_DECL float *bigvoice_GetBufferOut(struct bigvoice_CInstance *instance);
00826
00834 BIGVOICE_DECL unsigned int bigvoice_GetEstimatedDelay(struct bigvoice_CInstance *instance);
00835
00838 // -----
00845 // -----
00846 #ifdef _MSC_VER
00847 #define __BYTE_ORDER__ REG_DWORD
00848 #define __ORDER_LITTLE_ENDIAN__ REG_DWORD_LITTLE_ENDIAN
00849 #define __ORDER_BIG_ENDIAN__ REG_DWORD_BIG_ENDIAN
00850 #endif
00851
00855 enum bigvoice_SampleFormat {
00856     INVALID_FORMAT=0,
00857     S16_LE,
00858     S16_BE,
00859     S24_LE,
00860     S24_BE,
00861     S32_LE,
00862     S32_BE,
00863     F32_LE,
00864     F32_BE,
00865
00866     // Native definitions, only links to the physical ones
00867     #if __BYTE_ORDER__ == __ORDER_LITTLE_ENDIAN__
00868         S16_NATIVE=S16_LE,
00869         S24_NATIVE=S24_LE,
00870         S32_NATIVE=S32_LE,
00871         F32_NATIVE=F32_LE,
00872     #else
00873         S16_NATIVE=S16_BE,
00874         S24_NATIVE=S24_BE,
00875         S32_NATIVE=S32_BE,
00876         F32_NATIVE=F32_BE,
00877     #endif
00878 };
00879
00886 BIGVOICE_DECL const char *bigvoice_GetFormatName(const enum bigvoice_SampleFormat fmt);
00887
00894 BIGVOICE_DECL enum bigvoice_SampleFormat bigvoice_GetFormatFromName(const char *name);
00895
00902 BIGVOICE_DECL unsigned int bigvoice_GetBytesFromFormat(const enum bigvoice_SampleFormat fmt);
00903
00904
00911 BIGVOICE_DECL unsigned int bigvoice_GetMaxPacketFrame(struct bigvoice_CInstance *instance);
00912
00923 BIGVOICE_DECL unsigned int bigvoice_AddAudio(struct bigvoice_CInstance *instance, const uint8_t
*payload, unsigned int nFrame, enum bigvoice_SampleFormat fmt );
00924
00931 BIGVOICE_DECL unsigned int bigvoice_AddPadAudio(struct bigvoice_CInstance *instance);

```

```

00932
00939 BIGVOICE_DECL unsigned int bigvoice_GetOutputCount(struct bigvoice_CInstance *instance);
00940
00950 BIGVOICE_DECL unsigned int bigvoice_GetAudio(struct bigvoice_CInstance *instance, uint8_t *payload,
        unsigned int max_nFrame, enum bigvoice_SampleFormat fmt );
00951
00955 // -----
00959 // -----
00970 BIGVOICE_DECL void bigvoice_AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl, enum
        bigvoice_SampleFormat fmt);
00971
00982 BIGVOICE_DECL void bigvoice_AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl, enum
        bigvoice_SampleFormat fmt);
00983
00993 BIGVOICE_DECL void bigvoice_StereoToMono(const float *input, float *output, size_t nFrame);
00994
01004 BIGVOICE_DECL void bigvoice_MonoToStereo(const float *input, float *output, size_t nFrame);
01005
01016 BIGVOICE_DECL void bigvoice_StereoToMono_Planar(const float *inputL, const float *inputR, float
        *output, size_t nFrame);
01017
01028 BIGVOICE_DECL void bigvoice_MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
        size_t nFrame);
01029
01036 BIGVOICE_DECL void bigvoice_AudioMonoFromLiveStereo(const uint8_t *payload, float *output);
01037
01044 BIGVOICE_DECL void bigvoice_AudioMonoToLiveStereo(const float *input, uint8_t *payload);
01045
01048 // -----
01053 // -----
01057 struct bigvoice_CClientInstance;
01058
01067 BIGVOICE_DECL struct bigvoice_CClientInstance *bigvoice_NewClient(struct bigvoice_CInstance
        *instance);
01068
01074 BIGVOICE_DECL void bigvoice_DeleteClient(struct bigvoice_CClientInstance *client);
01075
01089 BIGVOICE_DECL const char *bigvoice_ProcessJson(struct bigvoice_CClientInstance *client, const char
        *json_str, int *need_save);
01090
01096 BIGVOICE_DECL void bigvoice_FreeJsonAnswer(const char *json_str);
01097
01109 BIGVOICE_DECL int bigvoice_SaveState(struct bigvoice_CInstance *instance);
01110
01406 #if BIGVOICE_HAS_WEBSERVER
01407 // -----
01415 // -----
01434 BIGVOICE_DECL uint64_t bigvoice_Webserver_tcp(unsigned int listenport, unsigned int listenport_secure,
        const char *socket_ip, unsigned int socket_port);
01435
01436 // -----
01456 BIGVOICE_DECL uint64_t bigvoice_Webserver_tcp2(unsigned int listenport, unsigned int
        listenport_secure, const char *socket_ip, unsigned int socket_port, const struct bigvoice_CParameters
        *parameters);
01457
01475 BIGVOICE_DECL uint64_t bigvoice_Webserver(unsigned int listenport, unsigned int listenport_secure,
        struct bigvoice_CInstance *instance);
01476
01486 BIGVOICE_DECL int bigvoice_Webserver_Stop(uint64_t id, int timeout_ms);
01487
01496 BIGVOICE_DECL int bigvoice_Webserver_Status(uint64_t id);
01497
01502 #define SOUND4_INVALID_WEBSERVER_ID ((uint64_t)-1)
01503
01507 #define SOUND4_WEBSERVER_HTTP_OK (1<<0)
01511 #define SOUND4_WEBSERVER_HTTPS_OK (1<<1)
01512
01514 #endif // BIGVOICE_HAS_WEBSERVER
01515
01516 #ifdef __cplusplus
01517 } // extern "C"
01518 #endif // __cplusplus

```

11.11 sound4.bigvoice.cl.hpp File Reference

C++ wrapper for library.

```

#include <string>
#include <exception>

```

```

#include <thread>
#include <functional>
#include <memory>
#include <array>
#include <assert.h>
#include <stdexcept>
#include <filesystem>
#include <string.h>
#include "sound4.bigvoice.cl.h"
#include <stringapiset.h>
#include <wchar.h>

```

Classes

- struct [sound4::bigvoice::helper::SampleFormat< T >](#)
Helper for Sample format types.
- struct [sound4::bigvoice::helper::SampleFormat< int16_t >](#)
- struct [sound4::bigvoice::helper::SampleFormat< int32_t >](#)
- struct [sound4::bigvoice::helper::SampleFormat< float >](#)
- class [sound4::bigvoice::CBus](#)
- class [sound4::bigvoice::CPresetLoader](#)
- class [sound4::bigvoice::CInstance](#)
- class [sound4::bigvoice::CInstance::CClient](#)

Namespaces

- [sound4](#)
- [sound4::bigvoice](#)
- [sound4::bigvoice::helper](#)

Macros

- `#define fs\_strdup wcsdup`
helper for OS dependant file name copy

Typedefs

- using [sound4::bigvoice::SampleFormat](#) = [bigvoice_SampleFormat](#)
- using [sound4::bigvoice::LogSeverity](#) = [bigvoice_LogSeverity](#)
- using [sound4::bigvoice::log_cb_t](#) = `std::function< void(LogSeverity, const std::string &)>`
- using [sound4::bigvoice::PresetChange_Kind](#) = [bigvoice_PresetChange_Kind](#)

Variables

- `const size_t sound4::bigvoice::ChunkFrames = BIGVOICE\_AUDIOFRAME\_COUNT`
- `const size_t sound4::bigvoice::InputChannels = BIGVOICE\_CHANNEL\_COUNT*BIGVOICE\_AUDIO\_INPUT\_COUNT`
- `const size_t sound4::bigvoice::OutputChannels = BIGVOICE\_CHANNEL\_COUNT*BIGVOICE\_AUDIO\_OUTPUT\_COUNT`
- `const size_t sound4::bigvoice::InputSampleSize = ChunkFrames*InputChannels`
- `const size_t sound4::bigvoice::OutputSampleSize = ChunkFrames*OutputChannels`
- `const size_t sound4::bigvoice::SampleRate = BIGVOICE\_SAMPLE\_RATE`

11.11.1 Macro Definition Documentation

11.11.1.1 fs_strdup

```
#define fs_strdup wcsdup
```

Definition at line 25 of file [sound4.bigvoice.cl.hpp](#).

11.12 sound4.bigvoice.cl.hpp

```
00001
00004 #pragma once
00005
00006 #include <string>
00007 #include <exception>
00008 #include <thread>
00009 #include <functional>
00010 #include <memory>
00011 #include <array>
00012 #include <assert.h>
00013 #include <stdexcept>
00014 #include <filesystem>
00015 #include <string.h>
00016
00017 #include "sound4.bigvoice.cl.h"
00018
00019
00020 // Filesystem path char type
00021 #ifdef _WIN32
00022     #include <stringapiset.h>
00023     #include <wchar.h>
00025     #define fs_strdup wcsdup
00026 #else // !_WIN32
00028     #define fs_strdup strdup
00029 #endif // !_WIN32
00030
00031 // bridge C callbacks to CPresetLoader
00032 extern "C" {
00033     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata);
00034     static void bigvoice_custom_reader_free(char *content, void* userdata);
00035     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata);
00036     static int bigvoice_custom_exists(const fs_char *filename, void* userdata);
00037     static fs_char** bigvoice_custom_getall(void* userdata);
00038     static void bigvoice_custom_getall_free(fs_char** all, void* userdata);
00039     static int bigvoice_custom_remove(const fs_char *filename, void* userdata);
00040     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00041 };
00042
00046 namespace sound4 {
00050 namespace bigvoice {
00051     static constexpr const char *process_name = "SOUND4 BIGVOICE.CL";
00052     static constexpr const char *process_shortcode = "bigvoice";
00053     using SampleFormat = bigvoice_SampleFormat;
00057     const size_t ChunkFrames=BIGVOICE_AUDIOFRAME_COUNT;
00061     const size_t InputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_INPUT_COUNT;
00065     const size_t OutputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_OUTPUT_COUNT;
00069     const size_t InputSampleSize=ChunkFrames*InputChannels;
00073     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00077     const size_t SampleRate=BIGVOICE_SAMPLE_RATE;
00078
00082     namespace helper {
00083         #ifdef _WIN32
00089         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00090             if (wstr.empty()) return {};
00091             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
NULL);
00092             std::string str(size, 0);
00093             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00094             return str;
00095         }
00101         static inline std::wstring UTF8ToWString(const std::string& str) {
00102             if (str.empty()) return std::wstring();
00103             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00104             std::wstring wstr(size, 0);
```

```

00105         MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00106         return wstr;
00107     }
00108 #endif // _WIN32
00109
00115     template<typename T>
00116     struct SampleFormat {
00117     };
00118     template<>
00119     struct SampleFormat<int16_t> {
00120         const bigvoice_SampleFormat format = S16_NATIVE;
00121     };
00122     template<>
00123     struct SampleFormat<int32_t> {
00124         const bigvoice_SampleFormat format = S32_NATIVE;
00125     };
00126     template<>
00127     struct SampleFormat<float> {
00128         const bigvoice_SampleFormat format = F32_NATIVE;
00129     };
00130
00131
00136     static inline void AudioConvertFrom(const uint8_t *payload, float *output, size_t nSpl,
bigvoice_SampleFormat fmt) {
00137         bigvoice_AudioConvertFrom(payload, output, nSpl, fmt);
00138     }
00139     static inline void AudioConvertTo(const float *input, uint8_t *payload, size_t nSpl,
bigvoice_SampleFormat fmt) {
00140         bigvoice_AudioConvertTo(input, payload, nSpl, fmt);
00141     }
00142     static inline void StereoToMono(const float *input, float *output, size_t nSpl) {
00143         bigvoice_StereoToMono(input, output, nSpl);
00144     }
00145     static inline void MonoToStereo(const float *input, float *output, size_t nSpl) {
00146         bigvoice_MonoToStereo(input, output, nSpl);
00147     }
00148     static inline void StereoToMono_Planar(const float *inputL, const float *inputR, float
*out, size_t nSpl) {
00149         bigvoice_StereoToMono_Planar(inputL, inputR, output, nSpl);
00150     }
00151     static inline void MonoToStereo_Planar(const float *input, float *outputL, float *outputR,
size_t nSpl) {
00152         bigvoice_MonoToStereo_Planar(input, outputL, outputR, nSpl);
00153     }
00154     static inline void AudioMonoFromLiveStereo(const uint8_t *payload, float *output) {
00155         bigvoice_AudioMonoFromLiveStereo(payload, output);
00156     }
00157     static inline void AudioMonoToLiveStereo(const float *input, uint8_t *payload) {
00158         bigvoice_AudioMonoToLiveStereo(input, payload);
00159     }
00160
00161 }; // namespace helper
00162
00168     static inline std::string GetVersion()
00169     {
00170         return std::string(bigvoice_GetVersion());
00171     }
00172
00183     static inline unsigned int GetChunkSizeInFrames()
00184     {
00185         return bigvoice_GetChunkSizeInFrames();
00186     }
00187
00196     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames()
00197     {
00198         std::vector<unsigned int> list;
00199         for (unsigned int* src=bigvoice_GetPossibleChunkSizeInFrames();*src;src++) {
00200             list.push_back(*src);
00201         }
00202         return list;
00203     }
00204
00212     static inline unsigned int GetChannelCount()
00213     {
00214         return bigvoice_GetChannelCount();
00215     }
00216
00225     static inline unsigned int GetAudioInputCount()
00226     {
00227         return bigvoice_GetAudioInputCount();
00228     }
00229
00238     static inline unsigned int GetAudioOutputCount()
00239     {
00240         return bigvoice_GetAudioOutputCount();
00241     }
00242

```

```

00250     static inline unsigned int GetSampleRate()
00251     {
00252         return bigvoice_GetSampleRate();
00253     }
00254
00260     static inline void SanityCheck(bool a_bCheckFrames = true)
00261     {
00262         // Do some sanity checks
00263         if (bigvoice_GetSampleRate() != BIGVOICE_SAMPLE_RATE) {
00264             throw std::runtime_error("Bad library sampling rate");
00265         }
00266         if (a_bCheckFrames && bigvoice_GetChunkSizeInFrames() != ChunkFrames) {
00267             throw std::runtime_error("Bad library frame size");
00268         }
00269         if (bigvoice_GetChannelCount() != BIGVOICE_CHANNEL_COUNT) {
00270             throw std::runtime_error("Bad library channel count");
00271         }
00272         if (bigvoice_GetAudioInputCount() != BIGVOICE_AUDIO_INPUT_COUNT) {
00273             throw std::runtime_error("Bad library input count");
00274         }
00275         if (bigvoice_GetAudioOutputCount() != BIGVOICE_AUDIO_OUTPUT_COUNT) {
00276             throw std::runtime_error("Bad library output count");
00277         }
00278     }
00279
00286     static inline std::string GetFormatName(const SampleFormat fmt)
00287     {
00288         return std::string(bigvoice_GetFormatName(fmt));
00289     }
00290
00297     static inline SampleFormat GetFormatFromName(const std::string& name)
00298     {
00299         return bigvoice_GetFormatFromName(name.c_str());
00300     }
00301
00308     static inline unsigned int GetBytesFromFormat(const SampleFormat fmt)
00309     {
00310         return bigvoice_GetBytesFromFormat(fmt);
00311     }
00312
00316     using LogSeverity = bigvoice_LogSeverity;
00320     using log_cb_t = std::function<void(LogSeverity, const std::string&);>;
00321
00325     static log_cb_t _log_cb;
00326     extern "C" {
00327         static inline void _log_cb_c(bigvoice_LogSeverity severity, const char *c_msg) {
00328             _log_cb(severity, std::string(c_msg));
00329         }
00330     }
00334     static inline void SetLogSeverity(LogSeverity severity)
00335     {
00336         bigvoice_SetLogSeverity(severity);
00337     }
00341     static inline void SetLoggerCallback(log_cb_t cb)
00342     {
00343         _log_cb=cb;
00344         bigvoice_SetLoggerCallback(&_log_cb_c);
00345     }
00346
00347 #if BIGVOICE_HAS_CLOUDBUS
00348     class CBus {
00349     public:
00350         CBus()
00351             : m_bus(bigvoice_NewBus()) {}
00352         ~CBus()
00353         {
00354             bigvoice_FreeBus(m_bus);
00355             m_bus = nullptr;
00356         }
00357         bigvoice_CBus *Get() const { return m_bus; }
00358     private:
00359         bigvoice_CBus *m_bus;
00360     };
00361 #endif // BIGVOICE_HAS_CLOUDBUS
00362
00367     class CPresetLoader {
00368     public:
00369         CPresetLoader() = default;
00370         virtual ~CPresetLoader() = default;
00371
00372         virtual bool IsReadOnly() = 0;
00373         virtual bool Exists(const std::filesystem::path &name) = 0;
00374         virtual bool Remove(const std::filesystem::path &name) = 0;
00375         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00376         virtual std::vector<std::filesystem::path> GetAll() = 0;
00377         virtual std::string Read(const std::filesystem::path &filename) = 0;
00378         virtual bool Write(const std::filesystem::path &filename, const std::string &content) = 0;

```

```

00379     };
00380     using PresetChange_Kind = bigvoice_PresetChange_Kind;
00381
00385     class CInstance {
00386     public:
00390         CInstance()
00391         {
00392             params = bigvoice_NewParameters();
00393             if (!params) throw std::bad_alloc();
00394         }
00395         // non-copyable
00396         CInstance( const CInstance& ) = delete;
00397         CInstance& operator=( const CInstance& ) = delete;
00401         ~CInstance()
00402         {
00403             Stop();
00404             if (params) bigvoice_FreeParameters(params);
00405         }
00409     bool IsOk() const { return instance!=nullptr; }
00417     void SetParam(const std::string& name, const std::string& value)
00418     {
00419         assert(params);
00420         bigvoice_SetParameter(params, name.c_str(), value.c_str());
00421     }
00427     std::string GetParam(const std::string& name)
00428     {
00429         assert(params);
00430         auto c_value=bigvoice_GetParameter(params, name.c_str());
00431         std::string ret(c_value);
00432         bigvoice_FreeParameterValue(c_value);
00433         return ret;
00434     }
00435
00445     void SetPresetManager(CPresetLoader *preset_manager)
00446     {
00447         if (!preset_manager) return;
00448         assert(params);
00449         bigvoice_SetPresetManager(params,
00450             bigvoice_custom_reader,
00451             bigvoice_custom_reader_free,
00452             bigvoice_custom_writer,
00453             bigvoice_custom_exists,
00454             bigvoice_custom_getall,
00455             bigvoice_custom_getall_free,
00456             bigvoice_custom_remove,
00457             bigvoice_custom_rename,
00458             preset_manager->IsReadOnly(),
00459             preset_manager
00460         );
00461     }
00462
00463 #if BIGVOICE_HAS_CLOUDBUS
00467     void SetBus(const CBus& bus) {
00468         assert(params);
00469         bigvoice_SetInstanceBus(params,bus.Get());
00470     }
00471 #endif // BIGVOICE_HAS_CLOUDBUS
00472
00487     bool Create(const std::string& LoginKey, const std::string& RadioName, const std::string&
AWS_Access_ID, const std::string& AWS_Access_Secret, const std::filesystem::path& save_path, int
json_port = 0, unsigned int frames_per_chunk=12)
00488     {
00489         assert(!instance);
00490         std::string l_save_path;
00491         if (!save_path.empty()) {
00492             l_save_path = save_path.u8string();
00493         }
00494         instance = bigvoice_InitProcess3(LoginKey.c_str(), RadioName.c_str(),
AWS_Access_ID.c_str(), AWS_Access_Secret.c_str(), l_save_path.c_str(), params, frames_per_chunk);
00495         if (!instance) return false;
00496         if (!init_metadata.empty()) {
00497             for (auto&& [key,value]: init_metadata) {
00498                 bigvoice_SetMetadata(instance, key.c_str(), value.c_str());
00499             }
00500             init_metadata.clear();
00501         }
00502         update_thread = std::thread([this, json_port]() {
00503             OnUpdateThreadStart();
00504             bigvoice_StartUpdateThread(instance, json_port);
00505             OnUpdateThreadStop();
00506         });
00507         int timeout=5000;
00508 #ifdef DEBUG
00509         timeout=60*1000; // For debugger pauses
00510 #endif
00511         if (bigvoice_WaitUpdateThreadReady(instance, timeout)<0) {
00512             return false;

```

```

00513         }
00514         return true;
00515     }
00516
00524     unsigned int GetChunkFrames() {
00525         if (instance) {
00526             return bigvoice_GetProcessChunkFrames(instance);
00527         }
00528         return 0;
00529     }
00530
00538     void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
change_kind)
00539     {
00540         bigvoice_PresetManager_InformChange(instance, relative_path.c_str(),
(bigvoice_PresetChange_Kind) change_kind);
00541     }
00542
00554     void SetMetadata(const std::string& key, const char* value)
00555     {
00556         if (instance) {
00557             bigvoice_SetMetadata(instance, key.c_str(), value);
00558         } else {
00559             init_metadata.push_back( {key, value} );
00560         }
00561     }
00562
00568     std::vector< std::tuple<std::string, std::string> > GetMetadataInfos()
00569     {
00570         std::vector< std::tuple<std::string, std::string> > values;
00571         const char** c_values = bigvoice_GetMetadataInfos(instance);
00572         if (c_values) {
00573             for (const char** c_value=c_values; *c_value; ) {
00574                 std::string key(*c_value);
00575                 c_value++;
00576                 if (*c_value) {
00577                     std::string descr(*c_value);
00578                     c_value++;
00579                     values.push_back( {key, descr} );
00580                 }
00581             }
00582             bigvoice_FreeMetadataInfos(instance, c_values);
00583         }
00584         return values;
00585     }
00586
00587     // Can be overloaded, called at start of UpdateThread in the thread
00588     virtual void OnUpdateThreadStart() {}
00589     // Can be overloaded to call at end of UpdateThread in the thread
00590     virtual void OnUpdateThreadStop() {}
00591
00592 #if BIGVOICE_HAS_WEBSERVER
00598     bool StartWebServer(int http_port, int https_port=0)
00599     {
00600         assert(instance);
00601         assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
00602         webserver = bigvoice_Webserver(http_port, https_port, instance);
00603         if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
00604             return false;
00605         }
00606         int web_status=bigvoice_Webserver_Status(webserver);
00607         if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
00608             return false;
00609         }
00610         if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
00611             return false;
00612         }
00613         return true;
00614     }
00620     void StopWebServer()
00621     {
00622         if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
00623             bigvoice_Webserver_Stop(webserver, 1000);
00624             webserver = SOUND4_INVALID_WEBSERVER_ID;
00625         }
00626     }
00627 #endif // BIGVOICE_HAS_WEBSERVER
00628
00635     int Terminate()
00636     {
00637         if (!instance) return 0;
00638         return bigvoice_TerminateProcess(instance);
00639     }
00640
00646     void Stop()
00647     {
00648         if (!instance) return;

```

```

00649 #if BIGVOICE_HAS_WEBSERVER
00650     StopWebServer();
00651 #endif // BIGVOICE_HAS_WEBSERVER
00652     if (update_thread.joinable()) {
00653         bigvoice_StopUpdateThread(instance); // this will exit the update thread
00654         update_thread.join();
00655     }
00656     bigvoice_ExitProcess(instance);
00657     instance=nullptr;
00658 }
00659
00665 unsigned int GetEstimatedDelay()
00666 {
00667     assert(instance);
00668     return bigvoice_GetEstimatedDelay(instance);
00669 }
00670
00681 std::array<float,InputSampleSize>& GetBufferIn()
00682 {
00683     assert(instance);
00684     float *buf=bigvoice_GetBufferIn(instance);
00685     return reinterpret_cast< std::array<float,InputSampleSize>& >(*buf);
00686 }
00687
00698 std::array<float,OutputSampleSize>& GetBufferOut()
00699 {
00700     assert(instance);
00701     float *buf= bigvoice_GetBufferOut(instance);
00702     return reinterpret_cast< std::array<float,OutputSampleSize>& >(*buf);
00703 }
00709 void ProcessAudio()
00710 {
00711     assert(instance);
00712     bigvoice_ProcessAudio(instance, bigvoice_GetBufferIn(instance),
bigvoice_GetBufferOut(instance));
00713 }
00727 // void ProcessAudio_Planar(const std::array< const std::array<const float,ChunkFrames>,
InputChannels>& input, std::array< std::array<float,ChunkFrames>, OutputChannels>& output)
00728 void ProcessAudio_Planar(float const * const *input, float * const *output)
00729 {
00730     assert(instance);
00731     bigvoice_ProcessAudio_Planar(instance, input, output);
00732 }
00733
00745 template<typename T>
00746 std::vector<T> ProcessAnyAudio(const std::vector<T> input)
00747 {
00748     assert(instance);
00749     std::vector<T> output;
00750     unsigned int out_offset=0;
00751     unsigned int in_offset=0;
00752     unsigned int todo = input.size();
00753     while (todo>0) {
00754         unsigned int left = AddAudio(&input[in_offset], todo);
00755         unsigned int out_avail = bigvoice_GetOutputCount(instance);
00756         output.resize(out_offset + out_avail);
00757         GetAudio(&output[out_offset], out_avail);
00758         out_offset+=out_avail;
00759         in_offset += todo-left;
00760         todo=left;
00761     }
00762     return output;
00763 }
00764
00770 class CClient {
00771 public:
00772     CClient(bigvoice_CInstance* instance)
00773     {
00774         assert(instance);
00775         client=bigvoice_NewClient(instance);
00776         if (!client) throw std::bad_alloc();
00777     }
00778     // non-copyable
00779     CClient( const CInstance& ) = delete;
00780     CClient& operator=( const CInstance& ) = delete;
00781     ~CClient()
00782     {
00783         if (client) {
00784             bigvoice_DeleteClient(client);
00785             client=nullptr;
00786         }
00787     }
00795     std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
00796     {
00797         assert(client);
00798         int need_save=0;
00799         const char *canswer = bigvoice_ProcessJson(client, request.c_str(),

```

```

&need_save);
00800         if (!canswer) return {};
00801         std::string answer(canswer);
00802         bigvoice_FreeJsonAnswer (canswer);
00803         if (NeedSave) {
00804             *NeedSave=(need_save!=0);
00805         }
00806         return answer;
00807     }
00808     private:
00809         bigvoice_CClientInstance *client = nullptr;
00810 };
00811 std::shared_ptr<CClient> NewClient()
00812 {
00820     assert(instance);
00821     return std::make_shared<CClient>(instance);
00822 }
00823
00824 protected:
00825     template<typename T>
00826     unsigned int AddAudio(const T* payload, unsigned int nFrame)
00827     {
00828         assert(instance);
00829         return bigvoice_AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
00830             helper::SampleFormat<T>::format);
00831     }
00832     template<typename T>
00833     unsigned int GetAudio(T* payload, unsigned int max_nFrame)
00834     {
00835         assert(instance);
00836         return bigvoice_GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
00837             helper::SampleFormat<T>::format);
00838     }
00839     private:
00840         bigvoice_CParameters* params = nullptr;
00841         bigvoice_CInstance* instance = nullptr;
00842         std::thread update_thread;
00843         std::vector< std::pair<std::string, std::string> > init_metadata;
00844 #if BIGVOICE_HAS_WEBSERVER
00845         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
00846 #endif // BIGVOICE_HAS_WEBSERVER
00847 };
00848 }; // namespace bigvoice
00849 }; // namespace sound4
00850
00851 // bridge C callbacks to CPresetLoader
00852 extern "C" {
00853     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata) {
00854         sound4::bigvoice::CPresetLoader
00855         *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
00856         std::string content=preset_loader->Read(filename);
00857         return strdup(content.c_str());
00858     }
00859     static void bigvoice_custom_reader_free(char *content, void* userdata) {
00860         if (!content) return;
00861         free(content);
00862     }
00863     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata) {
00864         sound4::bigvoice::CPresetLoader
00865         *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
00866         auto res=preset_loader->Write(filename,content);
00867         return res?0:-1;
00868     }
00869     static int bigvoice_custom_exists(const fs_char *filename, void* userdata) {
00870         sound4::bigvoice::CPresetLoader
00871         *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
00872         auto res=preset_loader->Exists(filename);
00873         return res?0:-1;
00874     }
00875     static fs_char** bigvoice_custom_getall(void* userdata) {
00876         sound4::bigvoice::CPresetLoader
00877         *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
00878         auto res=preset_loader->GetAll();
00879         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
00880         for (size_t n=0;n<res.size();n++) {
00881             all[n]=fs_strdup(res[n].c_str());
00882         }
00883         all[res.size()]=nullptr;
00884         return all;
00885     }
00886     static void bigvoice_custom_getall_free(fs_char** all, void* userdata) {
00887         if (!all) return;
00888         for (fs_char** one=all;*one;one++) {
00889             free(*one);
00890         }
00891     }

```

```

00887         free(all);
00888     }
00889     static int bigvoice_custom_remove(const fs_char *filename, void* userdata) {
00890         sound4::bigvoice::CPresetLoader
00891         *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
00892         auto res=preset_loader->Remove(filename);
00893         return res?0:-1;
00894     }
00895     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
00896         sound4::bigvoice::CPresetLoader
00897         *preset_loader=reinterpret_cast<sound4::bigvoice::CPresetLoader*>(userdata);
00898         auto res=preset_loader->Rename(from,to);
00899         return res?0:-1;
00900     };

```

11.13 sound4.bigvoice.cl_dyn.hpp File Reference

C++ wrapper for dynamic loading library.

```

#include <string>
#include <exception>
#include <thread>
#include <functional>
#include <memory>
#include <array>
#include <assert.h>
#include <stdexcept>
#include <filesystem>
#include <string.h>
#include <loaderapi.h>
#include "sound4.bigvoice.cl.h"
#include <stringapiset.h>
#include <wchar.h>

```

Classes

- class [sound4::bigvoice::dyn::helper::DynFuncHolder< T >](#)
Helper to wrap a function pointer.
- class [sound4::bigvoice::dyn::helper::CDynLoader](#)
Helper to load dynamic library.
- class [sound4::bigvoice::dyn::CDynLib](#)
Dynamic library interface.
- struct [sound4::bigvoice::dyn::helper::SampleFormat< T >](#)
Helper for Sample format types.
- struct [sound4::bigvoice::dyn::helper::SampleFormat< int16_t >](#)
- struct [sound4::bigvoice::dyn::helper::SampleFormat< int32_t >](#)
- struct [sound4::bigvoice::dyn::helper::SampleFormat< float >](#)
- class [sound4::bigvoice::dyn::CBus](#)
- class [sound4::bigvoice::dyn::CPresetLoader](#)
- class [sound4::bigvoice::dyn::CInstance](#)
- class [sound4::bigvoice::dyn::CInstance::CClient](#)

Namespaces

- [sound4](#)
- [sound4::bigvoice](#)
- [dyn](#)
- [sound4::bigvoice::dyn](#)
- [sound4::bigvoice::dyn::helper](#)

Macros

- `#define fs\_strdup wcsdup`
helper for OS dependant file name copy

Typedefs

- using [sound4::bigvoice::dyn::helper::DynLib_t](#) = HMODULE
- using [sound4::bigvoice::dyn::helper::DynFunc_t](#) = FARPROC
- using [sound4::bigvoice::dyn::SampleFormat](#) = [bigvoice_SampleFormat](#)
- using [sound4::bigvoice::dyn::LogSeverity](#) = [bigvoice_LogSeverity](#)
- using [sound4::bigvoice::dyn::log_cb_t](#) = std::function< void(LogSeverity, const std::string &);>
- using [sound4::bigvoice::dyn::PresetChange_Kind](#) = [bigvoice_PresetChange_Kind](#)

Variables

- const size_t [sound4::bigvoice::dyn::ChunkFrames](#) = [BIGVOICE_AUDIOFRAME_COUNT](#)
- const size_t [sound4::bigvoice::dyn::InputChannels](#) = [BIGVOICE_CHANNEL_COUNT](#)*[BIGVOICE_AUDIO_INPUT_COUNT](#)
- const size_t [sound4::bigvoice::dyn::OutputChannels](#) = [BIGVOICE_CHANNEL_COUNT](#)*[BIGVOICE_AUDIO_OUTPUT_COUNT](#)
- const size_t [sound4::bigvoice::dyn::InputSampleSize](#) = [ChunkFrames](#)*[InputChannels](#)
- const size_t [sound4::bigvoice::dyn::OutputSampleSize](#) = [ChunkFrames](#)*[OutputChannels](#)
- const size_t [sound4::bigvoice::dyn::SampleRate](#) = [BIGVOICE_SAMPLE_RATE](#)

11.13.1 Detailed Description

You need to load the dynamic library, and if it succeed you can use it.

Definition in file [sound4.bigvoice.cl_dyn.hpp](#).

11.13.2 Macro Definition Documentation

11.13.2.1 fs_strdup

```
#define fs_strdup wcsdup
```

Definition at line 36 of file [sound4.bigvoice.cl_dyn.hpp](#).

11.14 sound4.bigvoice.cl_dyn.hpp

```

00001
00007 #pragma once
00008
00009 #include <string>
00010 #include <exception>
00011 #include <thread>
00012 #include <functional>
00013 #include <memory>
00014 #include <array>
00015 #include <assert.h>
00016 #include <stdexcept>
00017 #include <filesystem>
00018 #include <string.h>
00019
00020 #ifdef __unix__
00021     #include <dlfcn.h>
00022 #elif defined(_WIN32)
00023     #include <loaderapi.h>
00024 #else
00025     #error "Unsupported OS"
00026 #endif
00027
00028 // Needed to have function declarations
00029 #include "sound4.bigvoice.cl.h"
00030
00031
00032 #ifdef _WIN32
00033     #include <stringapiset.h>
00034     #include <wchar.h>
00036     #define fs_strdup wcsdup
00037 #else // !_WIN32
00039     #define fs_strdup strdup
00040 #endif // !_WIN32
00041
00042 // bridge C callbacks to CPresetLoader
00043 extern "C" {
00044     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata);
00045     static void bigvoice_custom_reader_free(char *content, void* userdata);
00046     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata);
00047     static int bigvoice_custom_exists(const fs_char *filename, void* userdata);
00048     static fs_char** bigvoice_custom_getall(void* userdata);
00049     static void bigvoice_custom_getall_free(fs_char** all, void* userdata);
00050     static int bigvoice_custom_remove(const fs_char *filename, void* userdata);
00051     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata);
00052 };
00053
00057 namespace sound4 {
00061 namespace bigvoice {
00065 namespace dyn {
00066
00070     namespace helper {
00071         // Storage definition, depending on OS
00072         #ifdef __unix__
00073             using DynLib_t = void *;
00074             using DynFunc_t = void *;
00075         #else
00076             using DynLib_t = HMODULE;
00077             using DynFunc_t = FARPROC;
00078         #endif
00079
00085         template <typename T>
00086         class DynFuncHolder {
00087         private:
00088             DynFunc_t m_ptr = NULL;
00089         public:
00090             DynFuncHolder() = default;
00091             DynFuncHolder(DynFunc_t a_ptr) : m_ptr(a_ptr) {}
00092             operator bool() const { return (m_ptr!=NULL); }
00093             bool IsOk() const { return m_ptr!=NULL; }
00094             operator T* () const { return reinterpret_cast<T*>(m_ptr); }
00095         };
00096
00101         class CDynLoader {
00102         private:
00103             DynLib_t m_lib {};
00104         public:
00105             CDynLoader() = default;
00106             CDynLoader(CDynLoader&&) = default;
00107             // No copy allowed
00108             CDynLoader(const CDynLoader&) = delete;
00109             CDynLoader& operator=(CDynLoader const&) = delete;
00110             operator bool() const {
00111                 return (m_lib!=NULL);
00112             }
00113             bool IsOk() const {

```

```

00114         return (m_lib!=NULL);
00115     }
00116     #ifdef __unix__
00117     void Close() {
00118         if (IsOk()) {
00119             #ifndef __SANITIZE_ADDRESS__ // avoid unloading to keep call stack
00120                 dlclose(m_lib);
00121             #endif // __SANITIZE_ADDRESS__
00122             m_lib=NULL;
00123         }
00124     }
00125     bool Load(const std::filesystem::path& dynlib)
00126     {
00127         Close();
00128         m_lib=dlopen(dynlib.c_str(), RTLD_NOW|RTLD_LOCAL);
00129         return IsOk();
00130     }
00131     template <typename T>
00132     DynFuncHolder<T> GetSymbol(const std::string& name) {
00133         auto ptr=dlsym(m_lib, name.c_str());
00134         if (!ptr) {
00135             throw std::runtime_error("Missing function in library");
00136         }
00137         return DynFuncHolder<T>(ptr);
00138     }
00139     template <typename T>
00140     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00141         auto ptr=dlsym(m_lib, name.c_str());
00142         return DynFuncHolder<T>(ptr);
00143     }
00144     static std::filesystem::path GetThisLibraryPath(void) {
00145         static Dl_info info;
00146         if (! dladdr((void *) GetThisLibraryPath, & info)) return {};
00147         auto rp=realpath(info.dli_fname, NULL);
00148         std::filesystem::path p(rp);
00149         free(rp);
00150         return p;
00151     }
00152     #else
00153     bool Load(const std::filesystem::path& dynlib, uint32_t loadflags =
LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR|LOAD_LIBRARY_SEARCH_APPLICATION_DIR|LOAD_LIBRARY_SEARCH_USER_DIRS|LOAD_LIBRARY_SEARCH_
00154     {
00155         Close();
00156         m_lib=LoadLibraryExW(dynlib.c_str(), NULL, loadflags);
00157         return IsOk();
00158     }
00159     void Close() {
00160         if (IsOk()) {
00161             // NOTE: On Windows, FreeLibrary may create a lot of troubles, should use
FreeLibraryAndExitThread but...
00162             // So define SOUND4_CALL_FREELIBRARY before including this file if you
really want to unload the library dynamically
00163             #if defined(SOUND4_CALL_FREELIBRARYANDEXITTHREAD)
FreeLibraryAndExitThread(m_lib, 0);
00164             #elif defined(SOUND4_CALL_FREELIBRARY)
FreeLibrary(m_lib);
00165             #endif // SOUND4_CALL_FREELIBRARY
00166             m_lib=NULL;
00167         }
00168     }
00169     }
00170     template <typename T>
00171     DynFuncHolder<T> GetSymbol(const std::string& name) {
00172         auto ptr=GetProcAddress(m_lib, name.c_str());
00173         if (!ptr) {
00174             throw std::runtime_error("Missing function in library");
00175         }
00176         return DynFuncHolder<T>(ptr);
00177     }
00178     template <typename T>
00179     DynFuncHolder<T> GetSymbol_safe(const std::string& name) {
00180         auto ptr=GetProcAddress(m_lib, name.c_str());
00181         return DynFuncHolder<T>(ptr);
00182     }
00183     }
00184     static std::filesystem::path GetThisLibraryPath(void) {
00185         static wchar_t path[MAX_PATH]={};
00186         HMODULE hm = NULL;
00187         if (GetModuleHandleExW(GET_MODULE_HANDLE_EX_FLAG_FROM_ADDRESS |
GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT,
00188         (LPCWSTR) &GetThisLibraryPath, &hm))
00189         {
00190             if (GetModuleFileNameW(hm, path, MAX_PATH) > 0) {
00191                 return std::filesystem::path(path);
00192             }
00193         }
00194         return {};
00195     }
00196     #endif
00197 
```

```

00198         virtual ~CDynLoader() {
00199             Close();
00200         }
00201     }; // class CDynLoader
00202 }; // namespace helper
00203
00210 class CDynLib {
00211 private:
00212     helper::CDynLoader m_lib;
00213     bool m_bOK = false;
00214 public:
00215     CDynLib() = default;
00216
00217     // All those wrapped functions have the same signature as there C bigvoice_XXX
00218     helper::DynFuncHolder< decltype(bigvoice_GetVersion) > > GetVersion ;
00219     helper::DynFuncHolder< decltype(bigvoice_GetChunkSizeInFrames) > > GetChunkSizeInFrames ;
00220     helper::DynFuncHolder< decltype(bigvoice_GetChannelCount) > > GetChannelCount ;
00221     helper::DynFuncHolder< decltype(bigvoice_GetAudioInputCount) > > GetAudioInputCount ;
00222     helper::DynFuncHolder< decltype(bigvoice_GetAudioOutputCount) > > GetAudioOutputCount ;
00223     helper::DynFuncHolder< decltype(bigvoice_GetSampleRate) > > GetSampleRate ;
00224     helper::DynFuncHolder< decltype(bigvoice_SetLoggerCallback) > > SetLoggerCallback ;
00225     helper::DynFuncHolder< decltype(bigvoice_SetLogSeverity) > > SetLogSeverity ;
00226     helper::DynFuncHolder< decltype(bigvoice_NewParameters) > > NewParameters ;
00227     helper::DynFuncHolder< decltype(bigvoice_FreeParameters) > > FreeParameters ;
00228     helper::DynFuncHolder< decltype(bigvoice_SetParameter) > > SetParameter ;
00229     helper::DynFuncHolder< decltype(bigvoice_GetParameter) > > GetParameter ;
00230     helper::DynFuncHolder< decltype(bigvoice_FreeParameterValue) > > FreeParameterValue ;
00231     helper::DynFuncHolder< decltype(bigvoice_InitProcess) > > InitProcess ;
00232     helper::DynFuncHolder< decltype(bigvoice_InitProcess2) > > InitProcess2 ;
00233     helper::DynFuncHolder< decltype(bigvoice_TerminateProcess) > > TerminateProcess ;
00234     helper::DynFuncHolder< decltype(bigvoice_ExitProcess) > > ExitProcess ;
00235     helper::DynFuncHolder< decltype(bigvoice_StartUpdateThread) > > StartUpdateThread ;
00236     helper::DynFuncHolder< decltype(bigvoice_StopUpdateThread) > > StopUpdateThread ;
00237     helper::DynFuncHolder< decltype(bigvoice_WaitUpdateThreadReady) > > WaitUpdateThreadReady ;
00238     helper::DynFuncHolder< decltype(bigvoice_ProcessAudio) > > ProcessAudio ;
00239     helper::DynFuncHolder< decltype(bigvoice_ProcessAudio_Planar) > > ProcessAudio_Planar ;
00240     helper::DynFuncHolder< decltype(bigvoice_GetBufferIn) > > GetBufferIn ;
00241     helper::DynFuncHolder< decltype(bigvoice_GetBufferOut) > > GetBufferOut ;
00242     helper::DynFuncHolder< decltype(bigvoice_GetEstimatedDelay) > > GetEstimatedDelay ;
00243     helper::DynFuncHolder< decltype(bigvoice_GetFormatName) > > GetFormatName ;
00244     helper::DynFuncHolder< decltype(bigvoice_GetFormatFromName) > > GetFormatFromName ;
00245     helper::DynFuncHolder< decltype(bigvoice_GetBytesFromFormat) > > GetBytesFromFormat ;
00246     helper::DynFuncHolder< decltype(bigvoice_GetMaxPacketFrame) > > GetMaxPacketFrame ;
00247     helper::DynFuncHolder< decltype(bigvoice_AddAudio) > > AddAudio ;
00248     helper::DynFuncHolder< decltype(bigvoice_AddPadAudio) > > AddPadAudio ;
00249     helper::DynFuncHolder< decltype(bigvoice_GetOutputCount) > > GetOutputCount ;
00250     helper::DynFuncHolder< decltype(bigvoice_GetAudio) > > GetAudio ;
00251     helper::DynFuncHolder< decltype(bigvoice_AudioConvertFrom) > > AudioConvertFrom ;
00252     helper::DynFuncHolder< decltype(bigvoice_AudioConvertTo) > > AudioConvertTo ;
00253     helper::DynFuncHolder< decltype(bigvoice_StereoToMono) > > StereoToMono ;
00254     helper::DynFuncHolder< decltype(bigvoice_MonoToStereo) > > MonoToStereo ;
00255     helper::DynFuncHolder< decltype(bigvoice_AudioMonoFromLiveStereo) > > AudioMonoFromLiveStereo ;
00256     helper::DynFuncHolder< decltype(bigvoice_AudioMonoToLiveStereo) > > AudioMonoToLiveStereo ;
00257     helper::DynFuncHolder< decltype(bigvoice_NewClient) > > NewClient ;
00258     helper::DynFuncHolder< decltype(bigvoice_DeleteClient) > > DeleteClient ;
00259     helper::DynFuncHolder< decltype(bigvoice_ProcessJson) > > ProcessJson ;
00260     helper::DynFuncHolder< decltype(bigvoice_FreeJsonAnswer) > > FreeJsonAnswer ;
00261     helper::DynFuncHolder< decltype(bigvoice_SaveState) > > SaveState ;
00262 #if BIGVOICE_HAS_WEBSERVER
00263     helper::DynFuncHolder< decltype(bigvoice_Webserver_tcp) > > Webserver_tcp ;
00264     helper::DynFuncHolder< decltype(bigvoice_Webserver_tcp2) > > Webserver_tcp2 ;
00265     helper::DynFuncHolder< decltype(bigvoice_Webserver) > > Webserver ;
00266     helper::DynFuncHolder< decltype(bigvoice_Webserver_Stop) > > Webserver_Stop ;
00267     helper::DynFuncHolder< decltype(bigvoice_Webserver_Status) > > Webserver_Status ;
00268 #endif // BIGVOICE_HAS_WEBSERVER
00269
00270     // added 2023-03-22
00271     helper::DynFuncHolder< decltype(bigvoice_StereoToMono_Planar) > > StereoToMono_Planar ;
00272     helper::DynFuncHolder< decltype(bigvoice_MonoToStereo_Planar) > > MonoToStereo_Planar ;
00273
00274     // added 2023-05-22
00275     helper::DynFuncHolder< decltype(bigvoice_SetMetadata) > > SetMetadata ;
00276     helper::DynFuncHolder< decltype(bigvoice_GetMetadataInfos) > > GetMetadataInfos ;
00277     helper::DynFuncHolder< decltype(bigvoice_FreeMetadataInfos) > > FreeMetadataInfos ;
00278
00279     // added 2023-06-20
00280     helper::DynFuncHolder< decltype(bigvoice_SetPresetManager) > > SetPresetManager ;
00281 ;
00282     helper::DynFuncHolder< decltype(bigvoice_PresetManager_InformChange) > >
00283     PresetManager_InformChange ;
00284
00285     // added 2023-09-21
00286     helper::DynFuncHolder< decltype(bigvoice_GetPossibleChunkSizeInFrames) > >
00287     GetPossibleChunkSizeInFrames ;
00288     helper::DynFuncHolder< decltype(bigvoice_GetProcessChunkFrames) > >
00289     GetProcessChunkFrames ;
00290     helper::DynFuncHolder< decltype(bigvoice_InitProcess3) > > InitProcess3 ;

```

```

;
00287
00288 #if BIGVOICE_HAS_CLOUDBUS
00289     // added 2023-06-26
00290     helper::DynFuncHolder< decltype(bigvoice_NewBus)                ) > NewBus                ;
00291     helper::DynFuncHolder< decltype(bigvoice_FreeBus)              ) > FreeBus              ;
00292     helper::DynFuncHolder< decltype(bigvoice_SetInstanceBus)       ) > SetInstanceBus       ;
00293 #endif // BIGVOICE_HAS_CLOUDBUS
00294
00302     bool Load(const std::filesystem::path& path = std::filesystem::current_path()) {
00303         std::string filename;
00304         #ifdef __unix__
00305             filename="libsound4.bigvoice.cl.so";
00306         #else
00307             filename="sound4.bigvoice.cl.dll";
00308         #endif
00309         auto p = path;
00310         p.append(filename);
00311         m_lib.Load(p.native());
00312         if (!m_lib.IsOk()) {
00313             // Search in the same directory this code is in
00314             p =helper::CDynLoader::GetThisLibraryPath();
00315             p.append(filename);
00316             m_lib.Load(p.native());
00317             if (!m_lib.IsOk()) {
00318                 return false;
00319             }
00320         }
00321         // Load all C functions
00322         try {
00323             GetVersion                = m_lib.GetSymbol< decltype(bigvoice_GetVersion)                )
00324 >("bigvoice_GetVersion"                );
00325             GetChunkSizeInFrames      = m_lib.GetSymbol< decltype(bigvoice_GetChunkSizeInFrames      )
00326 >("bigvoice_GetChunkSizeInFrames"      );
00327             GetChannelCount           = m_lib.GetSymbol< decltype(bigvoice_GetChannelCount           )
00328 >("bigvoice_GetChannelCount"           );
00329             GetAudioInputCount        = m_lib.GetSymbol< decltype(bigvoice_GetAudioInputCount        )
00330 >("bigvoice_GetAudioInputCount"        );
00331             GetAudioOutputCount       = m_lib.GetSymbol< decltype(bigvoice_GetAudioOutputCount       )
00332 >("bigvoice_GetAudioOutputCount"       );
00333             GetSampleRate             = m_lib.GetSymbol< decltype(bigvoice_GetSampleRate             )
00334 >("bigvoice_GetSampleRate"             );
00335             SetLoggerCallback         = m_lib.GetSymbol< decltype(bigvoice_SetLoggerCallback         )
00336 >("bigvoice_SetLoggerCallback"         );
00337             SetLogSeverity            = m_lib.GetSymbol< decltype(bigvoice_SetLogSeverity            )
00338 >("bigvoice_SetLogSeverity"            );
00339             NewParameters             = m_lib.GetSymbol< decltype(bigvoice_NewParameters             )
00340 >("bigvoice_NewParameters"             );
00341             FreeParameters            = m_lib.GetSymbol< decltype(bigvoice_FreeParameters            )
00342 >("bigvoice_FreeParameters"            );
00343             SetParameter              = m_lib.GetSymbol< decltype(bigvoice_SetParameter              )
00344 >("bigvoice_SetParameter"              );
00345             GetParameter              = m_lib.GetSymbol< decltype(bigvoice_GetParameter              )
00346 >("bigvoice_GetParameter"              );
00347             FreeParameterValue        = m_lib.GetSymbol< decltype(bigvoice_FreeParameterValue        )
00348 >("bigvoice_FreeParameterValue"        );
00349             InitProcess               = m_lib.GetSymbol< decltype(bigvoice_InitProcess               )
00350 >("bigvoice_InitProcess"               );
00351             InitProcess2              = m_lib.GetSymbol< decltype(bigvoice_InitProcess2              )
00352 >("bigvoice_InitProcess2"              );
00353             TerminateProcess          = m_lib.GetSymbol< decltype(bigvoice_TerminateProcess          )
00354 >("bigvoice_TerminateProcess"          );
00355             ExitProcess               = m_lib.GetSymbol< decltype(bigvoice_ExitProcess               )
00356 >("bigvoice_ExitProcess"               );
00357             StartUpdateThread         = m_lib.GetSymbol< decltype(bigvoice_StartUpdateThread         )
00358 >("bigvoice_StartUpdateThread"         );
00359             StopUpdateThread          = m_lib.GetSymbol< decltype(bigvoice_StopUpdateThread          )
00360 >("bigvoice_StopUpdateThread"          );
00361             WaitUpdateThreadReady     = m_lib.GetSymbol< decltype(bigvoice_WaitUpdateThreadReady     )
00362 >("bigvoice_WaitUpdateThreadReady"     );
00363             ProcessAudio              = m_lib.GetSymbol< decltype(bigvoice_ProcessAudio              )
00364 >("bigvoice_ProcessAudio"              );
00365             ProcessAudio_Planar       = m_lib.GetSymbol< decltype(bigvoice_ProcessAudio_Planar       )
00366 >("bigvoice_ProcessAudio_Planar"       );
00367             GetBufferIn               = m_lib.GetSymbol< decltype(bigvoice_GetBufferIn               )
00368 >("bigvoice_GetBufferIn"               );
00369             GetBufferOut              = m_lib.GetSymbol< decltype(bigvoice_GetBufferOut              )
00370 >("bigvoice_GetBufferOut"              );
00371             GetEstimatedDelay         = m_lib.GetSymbol< decltype(bigvoice_GetEstimatedDelay         )
00372 >("bigvoice_GetEstimatedDelay"         );
00373             GetFormatName             = m_lib.GetSymbol< decltype(bigvoice_GetFormatName             )
00374 >("bigvoice_GetFormatName"             );
00375             GetFormatFromName         = m_lib.GetSymbol< decltype(bigvoice_GetFormatFromName         )
00376 >("bigvoice_GetFormatFromName"         );
00377             GetBytesFromFormat        = m_lib.GetSymbol< decltype(bigvoice_GetBytesFromFormat        )
00378 >("bigvoice_GetBytesFromFormat"        );
00379             GetMaxPacketFrame         = m_lib.GetSymbol< decltype(bigvoice_GetMaxPacketFrame         )
00380 >("bigvoice_GetMaxPacketFrame"         );

```

```

>("bigvoice_GetMaxPacketFrame"      );
00352     AddAudio                      = m_lib.GetSymbol< decltype(bigvoice_AddAudio)      )
>("bigvoice_AddAudio"                );
00353     AddPadAudio                   = m_lib.GetSymbol< decltype(bigvoice_AddPadAudio)    )
>("bigvoice_AddPadAudio"              );
00354     GetOutputCount                 = m_lib.GetSymbol< decltype(bigvoice_GetOutputCount)  )
>("bigvoice_GetOutputCount"           );
00355     GetAudio                       = m_lib.GetSymbol< decltype(bigvoice_GetAudio)        )
>("bigvoice_GetAudio"                 );
00356     AudioConvertFrom                = m_lib.GetSymbol< decltype(bigvoice_AudioConvertFrom) )
>("bigvoice_AudioConvertFrom"         );
00357     AudioConvertTo                  = m_lib.GetSymbol< decltype(bigvoice_AudioConvertTo)  )
>("bigvoice_AudioConvertTo"           );
00358     StereoToMono                    = m_lib.GetSymbol< decltype(bigvoice_StereoToMono)    )
>("bigvoice_StereoToMono"              );
00359     MonoToStereo                    = m_lib.GetSymbol< decltype(bigvoice_MonoToStereo)    )
>("bigvoice_MonoToStereo"              );
00360     AudioMonoFromLiveStereo         = m_lib.GetSymbol< decltype(bigvoice_AudioMonoFromLiveStereo) )
>("bigvoice_AudioMonoFromLiveStereo");
00361     AudioMonoToLiveStereo           = m_lib.GetSymbol< decltype(bigvoice_AudioMonoToLiveStereo) )
>("bigvoice_AudioMonoToLiveStereo" );
00362     NewClient                       = m_lib.GetSymbol< decltype(bigvoice_NewClient)       )
>("bigvoice_NewClient"                 );
00363     DeleteClient                    = m_lib.GetSymbol< decltype(bigvoice_DeleteClient)    )
>("bigvoice_DeleteClient"              );
00364     ProcessJson                     = m_lib.GetSymbol< decltype(bigvoice_ProcessJson)     )
>("bigvoice_ProcessJson"                );
00365     FreeJsonAnswer                  = m_lib.GetSymbol< decltype(bigvoice_FreeJsonAnswer)  )
>("bigvoice_FreeJsonAnswer"             );
00366     SaveState                       = m_lib.GetSymbol< decltype(bigvoice_SaveState)       )
>("bigvoice_SaveState"                 );
00367 #if BIGVOICE_HAS_WEBSERVER
00368     Webserver_tcp                    = m_lib.GetSymbol< decltype(bigvoice_Webserver_tcp)    )
>("bigvoice_Webserver_tcp"              );
00369     Webserver_tcp2                   = m_lib.GetSymbol< decltype(bigvoice_Webserver_tcp2)   )
>("bigvoice_Webserver_tcp2"             );
00370     Webserver                        = m_lib.GetSymbol< decltype(bigvoice_Webserver)       )
>("bigvoice_Webserver"                  );
00371     Webserver_Stop                   = m_lib.GetSymbol< decltype(bigvoice_Webserver_Stop)  )
>("bigvoice_Webserver_Stop"             );
00372     Webserver_Status                 = m_lib.GetSymbol< decltype(bigvoice_Webserver_Status) )
>("bigvoice_Webserver_Status"          );
00373 #endif // BIGVOICE_HAS_WEBSERVER
00374
00375     // added 2023-03-22
00376     StereoToMono_Planar              = m_lib.GetSymbol< decltype(bigvoice_StereoToMono_Planar) )
>("bigvoice_StereoToMono_Planar"        );
00377     MonoToStereo_Planar              = m_lib.GetSymbol< decltype(bigvoice_MonoToStereo_Planar) )
>("bigvoice_MonoToStereo_Planar"        );
00378
00379     } catch (std::runtime_error& ) {
00380         return false;
00381     }
00382     // C functions allowed to be missing
00383     try {
00384         // added 2023-05-22
00385         SetMetadata                   = m_lib.GetSymbol< decltype(bigvoice_SetMetadata)      )
>("bigvoice_SetMetadata"                 );
00386         GetMetadataInfos              = m_lib.GetSymbol< decltype(bigvoice_GetMetadataInfos)  )
>("bigvoice_GetMetadataInfos"            );
00387         FreeMetadataInfos             = m_lib.GetSymbol< decltype(bigvoice_FreeMetadataInfos) )
>("bigvoice_FreeMetadataInfos"          );
00388         // added 2023-06-20
00389         SetPresetManager              = m_lib.GetSymbol< decltype(bigvoice_SetPresetManager) )
>("bigvoice_SetPresetManager"            );
00390         PresetManager_InformChange    = m_lib.GetSymbol<
decltype(bigvoice_PresetManager_InformChange) >("bigvoice_PresetManager_InformChange" );
00391
00392         // added 2023-09-21
00393         GetPossibleChunkSizeInFrames = m_lib.GetSymbol<
decltype(bigvoice_GetPossibleChunkSizeInFrames) >("bigvoice_GetPossibleChunkSizeInFrames" );
00394         GetProcessChunkFrames         = m_lib.GetSymbol<
decltype(bigvoice_GetProcessChunkFrames) >("bigvoice_GetProcessChunkFrames" );
00395         InitProcess3                  = m_lib.GetSymbol< decltype(bigvoice_InitProcess3)
>("bigvoice_InitProcess3" );
00396
00397 #if BIGVOICE_HAS_CLOUDBUS
00398     // added 2023-06-26
00399     NewBus                           = m_lib.GetSymbol< decltype(bigvoice_NewBus)          )
>("bigvoice_NewBus"                      );
00400     FreeBus                          = m_lib.GetSymbol< decltype(bigvoice_FreeBus)         )
>("bigvoice_FreeBus"                    );
00401     SetInstanceBus                   = m_lib.GetSymbol< decltype(bigvoice_SetInstanceBus)  )
>("bigvoice_SetInstanceBus"              );
00402 #endif // BIGVOICE_HAS_CLOUDBUS
00403     } catch (std::runtime_error& ) {
00404         // Ignored, handler will take care

```

```

00405         }
00406
00407         m_bOK=true;
00408         return true;
00409     }
00416     bool IsOk() const {
00417         return m_bOK;
00418     }
00419
00420 };
00421
00422
00423
00424     static constexpr const char *process_name = "SOUND4 BIGVOICE.CL";
00425     static constexpr const char *process_shortname = "bigvoice";
00426     using SampleFormat = bigvoice_SampleFormat;
00430     const size_t ChunkFrames=BIGVOICE_AUDIOFRAME_COUNT;
00434     const size_t InputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_INPUT_COUNT;
00438     const size_t OutputChannels=BIGVOICE_CHANNEL_COUNT*BIGVOICE_AUDIO_OUTPUT_COUNT;
00442     const size_t InputSampleSize=ChunkFrames*InputChannels;
00446     const size_t OutputSampleSize=ChunkFrames*OutputChannels;
00450     const size_t SampleRate=BIGVOICE_SAMPLE_RATE;
00451
00455     namespace helper {
00456         #ifdef _WIN32
00462         static inline std::string WStringToUTF8(const std::wstring& wstr) {
00463             if (wstr.empty()) return {};
00464             int size = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
NULL);
00465             std::string str(size, 0);
00466             WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &str[0], size, NULL, NULL);
00467             return str;
00468         }
00474         static inline std::wstring UTF8ToWString(const std::string& str) {
00475             if (str.empty()) return std::wstring();
00476             int size = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
00477             std::wstring wstr(size, 0);
00478             MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstr[0], size);
00479             return wstr;
00480         }
00481         #endif // _WIN32
00482
00488         template<typename T>
00489         struct SampleFormat {
00490         };
00491         template<>
00492         struct SampleFormat<int16_t> {
00493             const bigvoice_SampleFormat format = S16_NATIVE;
00494         };
00495         template<>
00496         struct SampleFormat<int32_t> {
00497             const bigvoice_SampleFormat format = S32_NATIVE;
00498         };
00499         template<>
00500         struct SampleFormat<float> {
00501             const bigvoice_SampleFormat format = F32_NATIVE;
00502         };
00503
00504
00509     static inline void AudioConvertFrom(CDynLib& dynlib, const uint8_t *payload, float *output,
size_t nSpl, bigvoice_SampleFormat fmt) {
00510         dynlib.AudioConvertFrom(payload, output, nSpl, fmt);
00511     }
00512     static inline void AudioConvertTo(CDynLib& dynlib, const float *input, uint8_t *payload,
size_t nSpl, bigvoice_SampleFormat fmt) {
00513         dynlib.AudioConvertTo(input, payload, nSpl, fmt);
00514     }
00515     static inline void StereoToMono(CDynLib& dynlib, const float *input, float *output, size_t
nSpl) {
00516         dynlib.StereoToMono(input, output, nSpl);
00517     }
00518     static inline void MonoToStereo(CDynLib& dynlib, const float *input, float *output, size_t
nSpl) {
00519         dynlib.MonoToStereo(input, output, nSpl);
00520     }
00521     static inline void StereoToMono_Planar(CDynLib& dynlib, const float *inputL, const float
*inputR, float *output, size_t nSpl) {
00522         dynlib.StereoToMono_Planar(inputL, inputR, output, nSpl);
00523     }
00524     static inline void MonoToStereo_Planar(CDynLib& dynlib, const float *input, float *outputL,
float *outputR, size_t nSpl) {
00525         dynlib.MonoToStereo_Planar(input, outputL, outputR, nSpl);
00526     }
00527     static inline void AudioMonoFromLiveStereo(CDynLib& dynlib, const uint8_t *payload, float
*output) {
00528         dynlib.AudioMonoFromLiveStereo(payload, output);
00529     }

```

```

00530     static inline void AudioMonoToLiveStereo(CDynLib& dynlib, const float *input, uint8_t
    *payload) {
00531         dynlib.AudioMonoToLiveStereo(input, payload);
00532     }
00533
00534 }; // namespace helper
00535
00541     static inline std::string GetVersion(CDynLib& dynlib)
00542     {
00543         return std::string(dynlib.GetVersion());
00544     }
00545
00556     static inline unsigned int GetChunkSizeInFrames(CDynLib& dynlib)
00557     {
00558         return dynlib.GetChunkSizeInFrames();
00559     }
00560
00569     static inline std::vector<unsigned int> GetPossibleChunkSizeInFrames(CDynLib& dynlib)
00570     {
00571         std::vector<unsigned int> list;
00572         if (dynlib.GetPossibleChunkSizeInFrames() {
00573             for (unsigned int* src=dynlib.GetPossibleChunkSizeInFrames();*src;src++) {
00574                 list.push_back(*src);
00575             }
00576         } else {
00577             list.push_back(12);
00578         }
00579         return list;
00580     }
00581
00589     static inline unsigned int GetChannelCount(CDynLib& dynlib)
00590     {
00591         return dynlib.GetChannelCount();
00592     }
00593
00602     static inline unsigned int GetAudioInputCount(CDynLib& dynlib)
00603     {
00604         return dynlib.GetAudioInputCount();
00605     }
00606
00615     static inline unsigned int GetAudioOutputCount(CDynLib& dynlib)
00616     {
00617         return dynlib.GetAudioOutputCount();
00618     }
00619
00627     static inline unsigned int GetSampleRate(CDynLib& dynlib)
00628     {
00629         return dynlib.GetSampleRate();
00630     }
00631
00637     static inline void SanityCheck(CDynLib& dynlib, bool a_bCheckFrames = true)
00638     {
00639         // Do some sanity checks
00640         if (dynlib.GetSampleRate() != BIGVOICE_SAMPLE_RATE) {
00641             throw std::runtime_error("Bad library sampling rate");
00642         }
00643         if (a_bCheckFrames && dynlib.GetChunkSizeInFrames() != ChunkFrames) {
00644             throw std::runtime_error("Bad library frame size");
00645         }
00646         if (dynlib.GetChannelCount() != BIGVOICE_CHANNEL_COUNT) {
00647             throw std::runtime_error("Bad library channel count");
00648         }
00649         if (dynlib.GetAudioInputCount() != BIGVOICE_AUDIO_INPUT_COUNT) {
00650             throw std::runtime_error("Bad library input count");
00651         }
00652         if (dynlib.GetAudioOutputCount() != BIGVOICE_AUDIO_OUTPUT_COUNT) {
00653             throw std::runtime_error("Bad library output count");
00654         }
00655     }
00656
00663     static inline std::string GetFormatName(CDynLib& dynlib, const SampleFormat fmt)
00664     {
00665         return std::string(dynlib.GetFormatName(fmt));
00666     }
00667
00674     static inline SampleFormat GetFormatFromName(CDynLib& dynlib, const std::string& name)
00675     {
00676         return dynlib.GetFormatFromName(name.c_str());
00677     }
00678
00685     static inline unsigned int GetBytesFromFormat(CDynLib& dynlib, const SampleFormat fmt)
00686     {
00687         return dynlib.GetBytesFromFormat(fmt);
00688     }
00689
00693     using LogSeverity = bigvoice_LogSeverity;
00697     using log_cb_t = std::function<void(LogSeverity, const std::string&);>;

```

```

00698
00702     static log_cb_t _log_cb;
00703     extern "C" {
00704         static inline void _log_cb_c(bigvoice_LogSeverity severity, const char *c_msg) {
00705             _log_cb(severity, std::string(c_msg));
00706         }
00707     }
00711     static inline void SetLogSeverity(CDynLib& dynlib, LogSeverity severity)
00712     {
00713         dynlib.SetLogSeverity(severity);
00714     }
00718     static inline void SetLoggerCallback(CDynLib& dynlib, log_cb_t cb)
00719     {
00720         _log_cb=cb;
00721         dynlib.SetLoggerCallback(&_log_cb_c);
00722     }
00723
00724 #if BIGVOICE_HAS_CLOUDBUS
00725     class CBus {
00726     public:
00727         CBus(CDynLib& dynlib)
00728             : m_dynlib(dynlib)
00729         {
00730             if (m_dynlib.NewBus) {
00731                 m_bus=m_dynlib.NewBus();
00732             }
00733         }
00734         ~CBus()
00735         {
00736             if (m_bus && m_dynlib.FreeBus) {
00737                 m_dynlib.FreeBus(m_bus);
00738             }
00739             m_bus = nullptr;
00740         }
00741         bigvoice_CBus *Get() const { return m_bus; }
00742     private:
00743         CDynLib& m_dynlib;
00744         bigvoice_CBus *m_bus;
00745     };
00746 #endif // BIGVOICE_HAS_CLOUDBUS
00747
00752     class CPresetLoader {
00753     public:
00754         CPresetLoader() = default;
00755         virtual ~CPresetLoader() = default;
00756
00757         virtual bool IsReadOnly() = 0;
00758         virtual bool Exists(const std::filesystem::path &name) = 0;
00759         virtual bool Remove(const std::filesystem::path &name) = 0;
00760         virtual bool Rename(const std::filesystem::path &from, const std::filesystem::path &to) = 0;
00761         virtual std::vector<std::filesystem::path> GetAll() = 0;
00762         virtual std::string Read(const std::filesystem::path &filename) = 0;
00763         virtual bool Write(const std::filesystem::path &filename, const std::string &content) = 0;
00764     };
00765     using PresetChange_Kind = bigvoice_PresetChange_Kind;
00766
00770     class CInstance {
00771     public:
00772         CInstance(CDynLib& dynlib)
00773             : m_dynlib(dynlib)
00774         {
00775             params = m_dynlib.NewParameters();
00776             if (!params) throw std::bad_alloc();
00777         }
00778         // non-copyable
00779         CInstance(const CInstance& ) = delete;
00780         CInstance& operator=( const CInstance& ) = delete;
00781         ~CInstance()
00782         {
00783             Stop();
00784             if (params) m_dynlib.FreeParameters(params);
00785         }
00786     bool IsOk() const { return instance!=nullptr; }
00787     void SetParam(const std::string& name, const std::string& value)
00788     {
00789         assert(params);
00790         m_dynlib.SetParameter(params, name.c_str(), value.c_str());
00791     }
00792     std::string GetParam(const std::string& name)
00793     {
00794         assert(params);
00795         auto c_value=m_dynlib.GetParameter(params, name.c_str());
00796         std::string ret(c_value);
00797         m_dynlib.FreeParameterValue(c_value);
00798         return ret;
00799     }
00800 }
00801 #if BIGVOICE_HAS_CLOUDBUS

```

```

00825         void SetBus(const CBus& bus) {
00826             assert(params);
00827             if (m_dynlib.SetInstanceBus) {
00828                 m_dynlib.SetInstanceBus(params, bus.Get());
00829             }
00830         }
00831 #endif // BIGVOICE_HAS_CLOUDBUS
00832
00842         void SetPresetManager(CPresetLoader *preset_manager)
00843         {
00844             if (!preset_manager) return;
00845             if (!m_dynlib.SetPresetManager) return;
00846             assert(params);
00847             m_dynlib.SetPresetManager(params,
00848                 bigvoice_custom_reader,
00849                 bigvoice_custom_reader_free,
00850                 bigvoice_custom_writer,
00851                 bigvoice_custom_exists,
00852                 bigvoice_custom_getall,
00853                 bigvoice_custom_getall_free,
00854                 bigvoice_custom_remove,
00855                 bigvoice_custom_rename,
00856                 preset_manager->IsReadOnly(),
00857                 preset_manager
00858             );
00859         }
00860
00875         bool Create(const std::string& LoginKey, const std::string& RadioName, const std::string&
AWS_Access_ID, const std::string& AWS_Access_Secret, const std::filesystem::path& save_path, int
json_port = 0, unsigned int frames_per_chunk=12)
00876         {
00877             assert(!instance);
00878             std::string l_save_path;
00879             if (!save_path.empty()) {
00880                 l_save_path = save_path.u8string();
00881             }
00882             if (m_dynlib.InitProcess3) {
00883                 instance = m_dynlib.InitProcess3(LoginKey.c_str(), RadioName.c_str(),
AWS_Access_ID.c_str(), AWS_Access_Secret.c_str(), l_save_path.c_str(), params, frames_per_chunk);
00884             } else if (frames_per_chunk!=12) {
00885                 return false;
00886             } else {
00887                 instance = m_dynlib.InitProcess2(LoginKey.c_str(), RadioName.c_str(),
AWS_Access_ID.c_str(), AWS_Access_Secret.c_str(), l_save_path.c_str(), params);
00888             }
00889             if (!instance) return false;
00890             if (!init_metadata.empty()) {
00891                 for (auto&& [key,value]: init_metadata) {
00892                     m_dynlib.SetMetadata(instance, key.c_str(), value.c_str());
00893                 }
00894                 init_metadata.clear();
00895             }
00896             update_thread = std::thread([this, json_port]() {
00897                 OnUpdateThreadStart();
00898                 m_dynlib.StartUpdateThread(instance, json_port);
00899                 OnUpdateThreadStop();
00900             });
00901             int timeout=5000;
00902             #ifdef DEBUG
00903                 timeout=60*1000; // For debugger pauses
00904             #endif
00905             if (m_dynlib.WaitUpdateThreadReady(instance, timeout)<0) {
00906                 return false;
00907             }
00908             return true;
00909         }
00910
00918         unsigned int GetChunkFrames() {
00919             if (instance && m_dynlib.GetProcessChunkFrames) {
00920                 return m_dynlib.GetProcessChunkFrames(instance);
00921             }
00922             return 12;
00923         }
00924
00932         void PresetManager_InformChange(const std::filesystem::path& relative_path, PresetChange_Kind
change_kind)
00933         {
00934             if (m_dynlib.PresetManager_InformChange) {
00935                 m_dynlib.PresetManager_InformChange(instance, relative_path.c_str(),
(bigvoice_PresetChange_Kind) change_kind);
00936             }
00937         }
00938
00950         void SetMetadata(const std::string& key, const char* value)
00951         {
00952             if (m_dynlib.SetMetadata) {
00953                 if (instance) {

```

```

00954         m_dynlib.SetMetadata(instance, key.c_str(), value);
00955     } else {
00956         init_metadata.push_back( {key, value} );
00957     }
00958 }
00959 }
00960
00966 std::vector< std::tuple<std::string, std::string> > GetMetadataInfos()
00967 {
00968     std::vector< std::tuple<std::string, std::string> > values;
00969     if (m_dynlib.GetMetadataInfos) {
00970         const char** c_values = m_dynlib.GetMetadataInfos(instance);
00971         if (c_values) {
00972             for (const char** c_value=c_values; *c_value; ) {
00973                 std::string key(*c_value);
00974                 c_value++;
00975                 if (*c_value) {
00976                     std::string descr(*c_value);
00977                     c_value++;
00978                     values.push_back( {key, descr} );
00979                 }
00980             }
00981             m_dynlib.FreeMetadataInfos(instance, c_values);
00982         }
00983     }
00984     return values;
00985 }
00986
00987 // Can be overloaded, called at start of UpdateThread in the thread
00988 virtual void OnUpdateThreadStart() {}
00989 // Can be overloaded to call at end of UpdateThread in the thread
00990 virtual void OnUpdateThreadStop() {}
00991
00992 #if BIGVOICE_HAS_WEBSERVER
00998 bool StartWebServer(int http_port, int https_port=0)
00999 {
01000     assert(instance);
01001     assert(webserver==SOUND4_INVALID_WEBSERVER_ID);
01002     webserver = m_dynlib.Webserver(http_port, https_port, instance);
01003     if (webserver==SOUND4_INVALID_WEBSERVER_ID) {
01004         return false;
01005     }
01006     int web_status=m_dynlib.Webserver_Status(webserver);
01007     if (http_port && (web_status & SOUND4_WEBSERVER_HTTP_OK) == 0) {
01008         return false;
01009     }
01010     if (https_port && (web_status & SOUND4_WEBSERVER_HTTPS_OK) == 0) {
01011         return false;
01012     }
01013     return true;
01014 }
01020 void StopWebServer()
01021 {
01022     if (webserver != SOUND4_INVALID_WEBSERVER_ID) {
01023         m_dynlib.Webserver_Stop(webserver, 1000);
01024         webserver = SOUND4_INVALID_WEBSERVER_ID;
01025     }
01026 }
01027 #endif // BIGVOICE_HAS_WEBSERVER
01028
01035 int Terminate()
01036 {
01037     if (!instance) return 0;
01038     return m_dynlib.TerminateProcess(instance);
01039 }
01040
01046 void Stop()
01047 {
01048     if (!instance) return;
01049 #if BIGVOICE_HAS_WEBSERVER
01050     StopWebServer();
01051 #endif // BIGVOICE_HAS_WEBSERVER
01052     if (update_thread.joinable()) {
01053         m_dynlib.StopUpdateThread(instance); // this will exit the update thread
01054         update_thread.join();
01055     }
01056     m_dynlib.ExitProcess(instance);
01057     instance=nullptr;
01058 }
01059
01065 unsigned int GetEstimatedDelay()
01066 {
01067     assert(instance);
01068     return m_dynlib.GetEstimatedDelay(instance);
01069 }
01070
01081 std::array<float, InputSampleSize>& GetBufferIn()

```

```

01082         {
01083             assert(instance);
01084             float *buf=m_dynlib.GetBufferIn(instance);
01085             return reinterpret_cast< std::array<float, InputSampleSize>& >(*buf);
01086         }
01087     std::array<float, OutputSampleSize>& GetBufferOut ()
01088     {
01089         assert(instance);
01090         float *buf= m_dynlib.GetBufferOut(instance);
01091         return reinterpret_cast< std::array<float, OutputSampleSize>& >(*buf);
01092     }
01093 void ProcessAudio()
01094 {
01095     assert(instance);
01096     m_dynlib.ProcessAudio(instance, m_dynlib.GetBufferIn(instance),
01097 m_dynlib.GetBufferOut(instance));
01098 }
01099 // void ProcessAudio_Planar(const std::array< const std::array<const float, ChunkFrames>,
01100 InputChannels>& input, std::array< std::array<float, ChunkFrames>, OutputChannels>& output)
01101 void ProcessAudio_Planar(float const * const *input, float * const *output)
01102 {
01103     assert(instance);
01104     m_dynlib.ProcessAudio_Planar(instance, input, output);
01105 }
01106 template<typename T>
01107 std::vector<T> ProcessAnyAudio(const std::vector<T> input)
01108 {
01109     assert(instance);
01110     std::vector<T> output;
01111     unsigned int out_offset=0;
01112     unsigned int in_offset=0;
01113     unsigned int todo = input.size();
01114     while (todo>0) {
01115         unsigned int left = AddAudio(&input[in_offset], todo);
01116         unsigned int out_avail = m_dynlib.GetOutputCount(instance);
01117         output.resize(out_offset + out_avail);
01118         GetAudio(&output[out_offset], out_avail);
01119         out_offset+=out_avail;
01120         in_offset += todo-left;
01121         todo=left;
01122     }
01123     return output;
01124 }
01125 class CClient {
01126 public:
01127     CClient(bigvoice_CInstance* instance, CDynLib& dynlib)
01128     : m_dynlib(dynlib)
01129     {
01130         assert(instance);
01131         client=m_dynlib.NewClient(instance);
01132         if (!client) throw std::bad_alloc();
01133     }
01134     // non-copyable
01135     CClient( const CInstance& ) = delete;
01136     CClient& operator=( const CInstance& ) = delete;
01137     ~CClient()
01138     {
01139         if (client) {
01140             m_dynlib.DeleteClient(client);
01141             client=nullptr;
01142         }
01143     }
01144     std::string ProcessJson(const std::string &request, bool *NeedSave = nullptr)
01145     {
01146         assert(client);
01147         int need_save=0;
01148         const char *canswer = m_dynlib.ProcessJson(client, request.c_str(),
01149 &need_save);
01150         if (!canswer) return {};
01151         std::string answer(canswer);
01152         m_dynlib.FreeJsonAnswer (canswer);
01153         if (NeedSave) {
01154             *NeedSave=(need_save!=0);
01155         }
01156         return answer;
01157     }
01158 private:
01159     CDynLib& m_dynlib;
01160     bigvoice_CClientInstance *client = nullptr;
01161 };
01162 std::shared_ptr<CClient> NewClient()
01163 {
01164     assert(instance);
01165     return std::make_shared<CClient>(instance, m_dynlib);
01166 }

```

```

01224         }
01225     protected:
01226         template<typename T>
01227         unsigned int AddAudio(const T* payload, unsigned int nFrame)
01228     {
01229         {
01230             assert(instance);
01231             return m_dynlib.AddAudio(instance, reinterpret_cast<const T*>(payload), nFrame,
01232             helper::SampleFormat<T>::format);
01233         }
01234         template<typename T>
01235         unsigned int GetAudio(T* payload, unsigned int max_nFrame)
01236     {
01237         {
01238             assert(instance);
01239             return m_dynlib.GetAudio(instance, reinterpret_cast<T*>(payload), max_nFrame,
01240             helper::SampleFormat<T>::format);
01241         }
01242     private:
01243         CDynLib& m_dynlib;
01244         bigvoice_CParameters* params = nullptr;
01245         bigvoice_CInstance* instance = nullptr;
01246         std::thread update_thread;
01247         std::vector< std::pair<std::string, std::string> > init_metadata;
01248 #if BIGVOICE_HAS_WEBSERVER
01249         uint64_t webserver = SOUND4_INVALID_WEBSERVER_ID;
01250 #endif // BIGVOICE_HAS_WEBSERVER
01251     };
01252 }; // namespace dyn
01253 }; // namespace bigvoice
01254 }; // namespace sound4
01255
01256 // bridge C callbacks to CPresetLoader
01257 extern "C" {
01258     static char *bigvoice_custom_reader(const fs_char *filename, void* userdata) {
01259         sound4::bigvoice::dyn::CPresetLoader
01260         *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01261         std::string content=preset_loader->Read(filename);
01262         return strdup(content.c_str());
01263     }
01264     static void bigvoice_custom_reader_free(char *content, void* userdata) {
01265         if (!content) return;
01266         free(content);
01267     }
01268     static int bigvoice_custom_writer(const fs_char *filename, const char *content, void* userdata) {
01269         sound4::bigvoice::dyn::CPresetLoader
01270         *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01271         auto res=preset_loader->Write(filename,content);
01272         return res?0:-1;
01273     }
01274     static int bigvoice_custom_exists(const fs_char *filename, void* userdata) {
01275         sound4::bigvoice::dyn::CPresetLoader
01276         *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01277         auto res=preset_loader->Exists(filename);
01278         return res?0:-1;
01279     }
01280     static fs_char** bigvoice_custom_getall(void* userdata) {
01281         sound4::bigvoice::dyn::CPresetLoader
01282         *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01283         auto res=preset_loader->GetAll();
01284         fs_char**all=(fs_char**)malloc((res.size()+1)*sizeof(const fs_char*));
01285         for (size_t n=0;n<res.size();n++) {
01286             all[n]=fs_strdup(res[n].c_str());
01287         }
01288         all[res.size()]=nullptr;
01289         return all;
01290     }
01291     static void bigvoice_custom_getall_free(fs_char** all, void* userdata) {
01292         if (!all) return;
01293         for (fs_char** one=all;*one;one++) {
01294             free(*one);
01295         }
01296         free(all);
01297     }
01298     static int bigvoice_custom_remove(const fs_char *filename, void* userdata) {
01299         sound4::bigvoice::dyn::CPresetLoader
01300         *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01301         auto res=preset_loader->Remove(filename);
01302         return res?0:-1;
01303     }
01304     static int bigvoice_custom_rename(const fs_char *from, const fs_char *to, void* userdata) {
01305         sound4::bigvoice::dyn::CPresetLoader
01306         *preset_loader=reinterpret_cast<sound4::bigvoice::dyn::CPresetLoader*>(userdata);
01307         auto res=preset_loader->Rename(from,to);
01308         return res?0:-1;
01309     }
01310 }

```

```
01303     }  
01304  
01305 };
```


Index

- ~CBus
 - sound4::bigvoice::CBus, [79](#)
 - sound4::bigvoice::dyn::CBus, [80](#)
- ~CClient
 - sound4::bigvoice::CInstance::CClient, [81](#)
 - sound4::bigvoice::dyn::CInstance::CClient, [83](#)
- ~CDynLoader
 - sound4::bigvoice::dyn::helper::CDynLoader, [98](#)
- ~CInstance
 - sound4::bigvoice::CInstance, [102](#)
 - sound4::bigvoice::dyn::CInstance, [112](#)
- ~CPresetLoader
 - sound4::bigvoice::CPresetLoader, [121](#)
 - sound4::bigvoice::dyn::CPresetLoader, [123](#)
- ~MyPresetManager
 - MyPresetManager, [127](#)
- AddAudio
 - sound4::bigvoice::CInstance, [102](#)
 - sound4::bigvoice::dyn::CDynLib, [87](#)
 - sound4::bigvoice::dyn::CInstance, [112](#)
- AddPadAudio
 - sound4::bigvoice::dyn::CDynLib, [87](#)
- Audio convert helpers, [47](#)
 - bigvoice_AudioConvertFrom, [47](#)
 - bigvoice_AudioConvertTo, [48](#)
 - bigvoice_AudioMonoFromLiveStereo, [48](#)
 - bigvoice_AudioMonoToLiveStereo, [49](#)
 - bigvoice_MonoToStereo, [49](#)
 - bigvoice_MonoToStereo_Planar, [49](#)
 - bigvoice_StereoToMono, [50](#)
 - bigvoice_StereoToMono_Planar, [50](#)
- Audio managing (any audio), [42](#)
 - bigvoice_AddAudio, [43](#)
 - bigvoice_AddPadAudio, [43](#)
 - bigvoice_GetAudio, [44](#)
 - bigvoice_GetBytesFromFormat, [44](#)
 - bigvoice_GetFormatFromName, [45](#)
 - bigvoice_GetFormatName, [45](#)
 - bigvoice_GetMaxPacketFrame, [45](#)
 - bigvoice_GetOutputCount, [47](#)
 - bigvoice_SampleFormat, [42](#)
 - F32_BE, [43](#)
 - F32_LE, [43](#)
 - F32_NATIVE, [43](#)
 - INVALID_FORMAT, [43](#)
 - S16_BE, [43](#)
 - S16_LE, [43](#)
 - S16_NATIVE, [43](#)
 - S24_BE, [43](#)
 - S24_LE, [43](#)
 - S24_NATIVE, [43](#)
 - S32_BE, [43](#)
 - S32_LE, [43](#)
 - S32_NATIVE, [43](#)
- AudioConvertFrom
 - sound4::bigvoice::dyn::CDynLib, [87](#)
- AudioConvertTo
 - sound4::bigvoice::dyn::CDynLib, [87](#)
- AudioMonoFromLiveStereo
 - sound4::bigvoice::dyn::CDynLib, [87](#)
- AudioMonoToLiveStereo
 - sound4::bigvoice::dyn::CDynLib, [87](#)
- bigvoice_AddAudio
 - Audio managing (any audio), [43](#)
- bigvoice_AddPadAudio
 - Audio managing (any audio), [43](#)
- BIGVOICE_AUDIO_INPUT_COUNT
 - Global setup, [19](#)
- BIGVOICE_AUDIO_OUTPUT_COUNT
 - Global setup, [19](#)
- bigvoice_AudioConvertFrom
 - Audio convert helpers, [47](#)
- bigvoice_AudioConvertTo
 - Audio convert helpers, [48](#)
- BIGVOICE_AUDIOFRAME_COUNT
 - Global setup, [19](#)
- bigvoice_AudioMonoFromLiveStereo
 - Audio convert helpers, [48](#)
- bigvoice_AudioMonoToLiveStereo
 - Audio convert helpers, [49](#)
- bigvoice_CBus, [77](#)
- bigvoice_CClientInstance, [77](#)
- BIGVOICE_CHANNEL_COUNT
 - Global setup, [19](#)
- bigvoice_CInstance, [77](#)
- bigvoice_CParameters, [78](#)
- BIGVOICE_DECL
 - sound4.bigvoice.cl.h, [152](#)
- bigvoice_DeleteClient
 - Client's JSoN config interfacing, [51](#)
- bigvoice_ExitProcess
 - Process creation/destruction, [24](#)
- bigvoice_FreeBus
 - Instance's bus, [29](#)
- bigvoice_FreeJsonAnswer
 - Client's JSoN config interfacing, [52](#)
- bigvoice_FreeMetadataInfos
 - Metadata Management, [36](#)

- bigvoice_FreeParameters
 - Process creation/destruction, [25](#)
- bigvoice_FreeParameterValue
 - Process creation/destruction, [25](#)
- bigvoice_GetAudio
 - Audio managing (any audio), [44](#)
- bigvoice_GetAudioInputCount
 - Global setup, [21](#)
- bigvoice_GetAudioOutputCount
 - Global setup, [21](#)
- bigvoice_GetBufferIn
 - Direct processing, [39](#)
- bigvoice_GetBufferOut
 - Direct processing, [40](#)
- bigvoice_GetBytesFromFormat
 - Audio managing (any audio), [44](#)
- bigvoice_GetChannelCount
 - Global setup, [21](#)
- bigvoice_GetChunkSizeInFrames
 - Global setup, [22](#)
- bigvoice_GetEstimatedDelay
 - Direct processing, [40](#)
- bigvoice_GetFormatFromName
 - Audio managing (any audio), [45](#)
- bigvoice_GetFormatName
 - Audio managing (any audio), [45](#)
- bigvoice_GetMaxPacketFrame
 - Audio managing (any audio), [45](#)
- bigvoice_GetMetadataInfos
 - Metadata Management, [37](#)
- bigvoice_GetOutputCount
 - Audio managing (any audio), [47](#)
- bigvoice_GetParameter
 - Process creation/destruction, [25](#)
- bigvoice_GetPossibleChunkSizeInFrames
 - Global setup, [22](#)
- bigvoice_GetProcessChunkFrames
 - Process creation/destruction, [25](#)
- bigvoice_GetSampleRate
 - Global setup, [22](#)
- bigvoice_GetVersion
 - Global setup, [23](#)
- BIGVOICE_HAS_CLOUDBUS
 - sound4.bigvoice.cl.h, [152](#)
- BIGVOICE_HAS_WEBSEVER
 - sound4.bigvoice.cl.h, [152](#)
- bigvoice_InitProcess
 - Process creation/destruction, [26](#)
- bigvoice_InitProcess2
 - Process creation/destruction, [26](#)
- bigvoice_InitProcess3
 - Process creation/destruction, [26](#)
- bigvoice_loggerfn
 - Global setup, [20](#)
- bigvoice_LogSeverity
 - Global setup, [20](#)
- bigvoice_MonoToStereo
 - Audio convert helpers, [49](#)
- bigvoice_MonoToStereo_Planar
 - Audio convert helpers, [49](#)
- bigvoice_NewBus
 - Instance's bus, [29](#)
- bigvoice_NewClient
 - Client's JSon config interfacing, [52](#)
- bigvoice_NewParameters
 - Process creation/destruction, [28](#)
- bigvoice_PresetChange_Kind
 - Custom preset managagement, [35](#)
- bigvoice_PresetManager_InformChange
 - Custom preset managagement, [35](#)
- bigvoice_ProcessAudio
 - Direct processing, [40](#)
- bigvoice_ProcessAudio_Planar
 - Direct processing, [41](#)
- bigvoice_ProcessJson
 - Client's JSon config interfacing, [52](#)
- BIGVOICE_SAMPLE_RATE
 - Global setup, [20](#)
- bigvoice_SampleFormat
 - Audio managing (any audio), [42](#)
- bigvoice_SaveState
 - Client's JSon config interfacing, [53](#)
- bigvoice_SetInstanceBus
 - Instance's bus, [30](#)
- bigvoice_SetLoggerCallback
 - Global setup, [23](#)
- bigvoice_SetLogSeverity
 - Global setup, [23](#)
- bigvoice_SetMetadata
 - Metadata Management, [37](#)
- bigvoice_SetParameter
 - Process creation/destruction, [28](#)
- bigvoice_SetPresetManager
 - Custom preset managagement, [35](#)
- bigvoice_StartUpdateThread
 - Update Management, [38](#)
- bigvoice_StereoToMono
 - Audio convert helpers, [50](#)
- bigvoice_StereoToMono_Planar
 - Audio convert helpers, [50](#)
- bigvoice_StopUpdateThread
 - Update Management, [38](#)
- bigvoice_storage_exists
 - Custom preset managagement, [31](#)
- bigvoice_storage_getall
 - Custom preset managagement, [32](#)
- bigvoice_storage_getall_free
 - Custom preset managagement, [32](#)
- bigvoice_storage_reader
 - Custom preset managagement, [32](#)
- bigvoice_storage_reader_free
 - Custom preset managagement, [33](#)
- bigvoice_storage_remove
 - Custom preset managagement, [33](#)
- bigvoice_storage_rename
 - Custom preset managagement, [33](#)

- bigvoice_storage_writer
 - Custom preset management, [34](#)
- bigvoice_TerminateProcess
 - Process creation/destruction, [28](#)
- bigvoice_WaitUpdateThreadReady
 - Update Management, [39](#)
- bigvoice_Webserver
 - Web server, [61](#)
- bigvoice_Webserver_Status
 - Web server, [62](#)
- bigvoice_Webserver_Stop
 - Web server, [62](#)
- bigvoice_Webserver_tcp
 - Web server, [63](#)
- bigvoice_Webserver_tcp2
 - Web server, [63](#)
- CBus
 - sound4::bigvoice::CBus, [78](#)
 - sound4::bigvoice::dyn::CBus, [80](#)
- CClient
 - sound4::bigvoice::CInstance::CClient, [81](#)
 - sound4::bigvoice::dyn::CInstance::CClient, [83](#)
- CDynLib
 - sound4::bigvoice::dyn::CDynLib, [86](#)
- CDynLoader
 - sound4::bigvoice::dyn::helper::CDynLoader, [98](#)
- change_kind_created
 - Custom preset management, [35](#)
- change_kind_deleted
 - Custom preset management, [35](#)
- change_kind_modified
 - Custom preset management, [35](#)
- CHANNEL_COUNT
 - sound4.bigvoice.cl-pipe.cpp, [144](#)
- CHUNK_SIZE
 - sound4.bigvoice.cl-pipe.cpp, [145](#)
- ChunkFrames
 - sound4::bigvoice, [71](#)
 - sound4::bigvoice::dyn, [74](#)
- CInstance
 - sound4::bigvoice::CInstance, [101](#), [102](#)
 - sound4::bigvoice::dyn::CInstance, [112](#)
- Client's JSon config interfacing, [51](#)
 - bigvoice_DeleteClient, [51](#)
 - bigvoice_FreeJsonAnswer, [52](#)
 - bigvoice_NewClient, [52](#)
 - bigvoice_ProcessJson, [52](#)
 - bigvoice_SaveState, [53](#)
- Close
 - sound4::bigvoice::dyn::helper::CDynLoader, [98](#)
- CPresetLoader
 - sound4::bigvoice::CPresetLoader, [121](#)
 - sound4::bigvoice::dyn::CPresetLoader, [123](#)
- Create
 - sound4::bigvoice::CInstance, [102](#)
 - sound4::bigvoice::dyn::CInstance, [112](#)
- Custom preset management, [30](#)
 - bigvoice_PresetChange_Kind, [35](#)
 - bigvoice_PresetManager_InformChange, [35](#)
 - bigvoice_SetPresetManager, [35](#)
 - bigvoice_storage_exists, [31](#)
 - bigvoice_storage_getall, [32](#)
 - bigvoice_storage_getall_free, [32](#)
 - bigvoice_storage_reader, [32](#)
 - bigvoice_storage_reader_free, [33](#)
 - bigvoice_storage_remove, [33](#)
 - bigvoice_storage_rename, [33](#)
 - bigvoice_storage_writer, [34](#)
 - change_kind_created, [35](#)
 - change_kind_deleted, [35](#)
 - change_kind_modified, [35](#)
 - fs_char, [34](#)
- DeleteClient
 - sound4::bigvoice::dyn::CDynLib, [88](#)
- Direct processing, [39](#)
 - bigvoice_GetBufferIn, [39](#)
 - bigvoice_GetBufferOut, [40](#)
 - bigvoice_GetEstimatedDelay, [40](#)
 - bigvoice_ProcessAudio, [40](#)
 - bigvoice_ProcessAudio_Planar, [41](#)
- documentation.h, [135](#)
- dyn, [69](#)
- DynFunc_t
 - sound4::bigvoice::dyn::helper, [75](#)
- DynFuncHolder
 - sound4::bigvoice::dyn::helper::DynFuncHolder< T
>, [125](#)
- DynLib_t
 - sound4::bigvoice::dyn::helper, [75](#)
- Environment variables and parameters, [17](#)
- error
 - Global setup, [21](#)
- Exists
 - MyPresetManager, [127](#)
 - sound4::bigvoice::CPresetLoader, [121](#)
 - sound4::bigvoice::dyn::CPresetLoader, [123](#)
- ExitProcess
 - sound4::bigvoice::dyn::CDynLib, [88](#)
- F32_BE
 - Audio managing (any audio), [43](#)
- F32_LE
 - Audio managing (any audio), [43](#)
- F32_NATIVE
 - Audio managing (any audio), [43](#)
- fatal
 - Global setup, [21](#)
- format
 - sound4::bigvoice::dyn::helper::SampleFormat< float >, [130](#)
 - sound4::bigvoice::dyn::helper::SampleFormat< int16_t >, [132](#)
 - sound4::bigvoice::dyn::helper::SampleFormat< int32_t >, [133](#)

- sound4::bigvoice::helper::SampleFormat< float >, 131
 - sound4::bigvoice::helper::SampleFormat< int16_t >, 132
 - sound4::bigvoice::helper::SampleFormat< int32_t >, 134
- FreeBus
 - sound4::bigvoice::dyn::CDynLib, 88
- FreeJsonAnswer
 - sound4::bigvoice::dyn::CDynLib, 88
- FreeMetadataInfos
 - sound4::bigvoice::dyn::CDynLib, 88
- FreeParameters
 - sound4::bigvoice::dyn::CDynLib, 88
- FreeParameterValue
 - sound4::bigvoice::dyn::CDynLib, 89
- fs_char
 - Custom preset management, 34
- fs_strdup
 - sound4.bigvoice.cl.hpp, 158
 - sound4.bigvoice.cl_dyn.hpp, 166
- Get
 - sound4::bigvoice::CBus, 79
 - sound4::bigvoice::dyn::CBus, 80
- GetAll
 - MyPresetManager, 127
 - sound4::bigvoice::CPresetLoader, 121
 - sound4::bigvoice::dyn::CPresetLoader, 124
- GetAudio
 - sound4::bigvoice::CInstance, 103
 - sound4::bigvoice::dyn::CDynLib, 89
 - sound4::bigvoice::dyn::CInstance, 113
- GetAudioInputCount
 - sound4::bigvoice::dyn::CDynLib, 89
- GetAudioOutputCount
 - sound4::bigvoice::dyn::CDynLib, 89
- GetBufferIn
 - sound4::bigvoice::CInstance, 103
 - sound4::bigvoice::dyn::CDynLib, 89
 - sound4::bigvoice::dyn::CInstance, 113
- GetBufferOut
 - sound4::bigvoice::CInstance, 103
 - sound4::bigvoice::dyn::CDynLib, 89
 - sound4::bigvoice::dyn::CInstance, 113
- GetBytesFromFormat
 - sound4::bigvoice::dyn::CDynLib, 90
- GetChannelCount
 - sound4::bigvoice::dyn::CDynLib, 90
- GetChunkFrames
 - sound4::bigvoice::CInstance, 104
 - sound4::bigvoice::dyn::CInstance, 114
- GetChunkSizeInFrames
 - sound4::bigvoice::dyn::CDynLib, 90
- GetEnv
 - sound4.bigvoice.cl-pipe-cxx.cpp, 135
 - sound4.bigvoice.cl-pipe-dyn.cpp, 139
- GetEstimatedDelay
 - sound4::bigvoice::CInstance, 104
- sound4::bigvoice::dyn::CDynLib, 90
 - sound4::bigvoice::dyn::CInstance, 114
- GetFormatFromName
 - sound4::bigvoice::dyn::CDynLib, 90
- GetFormatName
 - sound4::bigvoice::dyn::CDynLib, 90
- GetMaxPacketFrame
 - sound4::bigvoice::dyn::CDynLib, 91
- GetMetadataInfos
 - sound4::bigvoice::CInstance, 104
 - sound4::bigvoice::dyn::CDynLib, 91
 - sound4::bigvoice::dyn::CInstance, 114
- GetOutputCount
 - sound4::bigvoice::dyn::CDynLib, 91
- GetParam
 - sound4::bigvoice::CInstance, 105
 - sound4::bigvoice::dyn::CInstance, 115
- GetParameter
 - sound4::bigvoice::dyn::CDynLib, 91
- GetPossibleChunkSizeInFrames
 - sound4::bigvoice::dyn::CDynLib, 91
- GetProcessChunkFrames
 - sound4::bigvoice::dyn::CDynLib, 91
- GetSampleRate
 - sound4::bigvoice::dyn::CDynLib, 92
- GetSymbol
 - sound4::bigvoice::dyn::helper::CDynLoader, 99
- GetSymbol_safe
 - sound4::bigvoice::dyn::helper::CDynLoader, 99
- GetThisLibraryPath
 - sound4::bigvoice::dyn::helper::CDynLoader, 99
- GetVersion
 - sound4::bigvoice::dyn::CDynLib, 92
- Global setup, 18
 - BIGVOICE_AUDIO_INPUT_COUNT, 19
 - BIGVOICE_AUDIO_OUTPUT_COUNT, 19
 - BIGVOICE_AUDIOFRAME_COUNT, 19
 - BIGVOICE_CHANNEL_COUNT, 19
 - bigvoice_GetAudioInputCount, 21
 - bigvoice_GetAudioOutputCount, 21
 - bigvoice_GetChannelCount, 21
 - bigvoice_GetChunkSizeInFrames, 22
 - bigvoice_GetPossibleChunkSizeInFrames, 22
 - bigvoice_GetSampleRate, 22
 - bigvoice_GetVersion, 23
 - bigvoice_loggerfn, 20
 - bigvoice_LogSeverity, 20
 - BIGVOICE_SAMPLE_RATE, 20
 - bigvoice_SetLoggerCallback, 23
 - bigvoice_SetLogSeverity, 23
 - error, 21
 - fatal, 21
 - info, 21
 - none, 21
 - verbose, 21
 - verbose2, 21
 - verbose3, 21
 - verbose4, 21

- verbose5, 21
- warning, 21
- info
 - Global setup, 21
- InitProcess
 - sound4::bigvoice::dyn::CDynLib, 92
- InitProcess2
 - sound4::bigvoice::dyn::CDynLib, 92
- InitProcess3
 - sound4::bigvoice::dyn::CDynLib, 92
- InputChannels
 - sound4::bigvoice, 71
 - sound4::bigvoice::dyn, 74
- InputSampleSize
 - sound4::bigvoice, 71
 - sound4::bigvoice::dyn, 74
- Instance's bus, 29
 - bigvoice_FreeBus, 29
 - bigvoice_NewBus, 29
 - bigvoice_SetInstanceBus, 30
- INVALID_FORMAT
 - Audio managing (any audio), 43
- IsOk
 - sound4::bigvoice::CInstance, 105
 - sound4::bigvoice::dyn::CDynLib, 86
 - sound4::bigvoice::dyn::CInstance, 115
 - sound4::bigvoice::dyn::helper::CDynLoader, 99
 - sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 126
- IsReadOnly
 - MyPresetManager, 128
 - sound4::bigvoice::CPresetLoader, 122
 - sound4::bigvoice::dyn::CPresetLoader, 124
- JSON syntax, 53
- JSON syntax for internal setup, 56
- Load
 - sound4::bigvoice::dyn::CDynLib, 86
 - sound4::bigvoice::dyn::helper::CDynLoader, 99
- log_cb_t
 - sound4::bigvoice, 70
 - sound4::bigvoice::dyn, 73
- LogSeverity
 - sound4::bigvoice, 70
 - sound4::bigvoice::dyn, 73
- main
 - sound4.bigvoice.cl-pipe-cxx.cpp, 136
 - sound4.bigvoice.cl-pipe-dyn.cpp, 139
 - sound4.bigvoice.cl-pipe.cpp, 145
- Metadata Management, 36
 - bigvoice_FreeMetadataInfos, 36
 - bigvoice_GetMetadataInfos, 37
 - bigvoice_SetMetadata, 37
- MonoToStereo
 - sound4::bigvoice::dyn::CDynLib, 92
- MonoToStereo_Planar
 - sound4::bigvoice::dyn::CDynLib, 93
- MyLogger
 - sound4.bigvoice.cl-pipe-cxx.cpp, 136
 - sound4.bigvoice.cl-pipe-dyn.cpp, 140
 - sound4.bigvoice.cl-pipe.cpp, 145
- MyPresetManager, 126
 - ~MyPresetManager, 127
 - Exists, 127
 - GetAll, 127
 - IsReadOnly, 128
 - MyPresetManager, 127
 - Read, 128
 - Remove, 128
 - Rename, 128
 - SetupPath, 128
 - Write, 129
- NewBus
 - sound4::bigvoice::dyn::CDynLib, 93
- NewClient
 - sound4::bigvoice::CInstance, 105
 - sound4::bigvoice::dyn::CDynLib, 93
 - sound4::bigvoice::dyn::CInstance, 115
- NewParameters
 - sound4::bigvoice::dyn::CDynLib, 93
- none
 - Global setup, 21
- OnUpdateThreadStart
 - sound4::bigvoice::CInstance, 105
 - sound4::bigvoice::dyn::CInstance, 115
- OnUpdateThreadStop
 - sound4::bigvoice::CInstance, 106
 - sound4::bigvoice::dyn::CInstance, 116
- operator bool
 - sound4::bigvoice::dyn::helper::CDynLoader, 100
 - sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 126
- operator T*
 - sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 126
- operator=
 - sound4::bigvoice::CInstance, 106
 - sound4::bigvoice::CInstance::CClient, 81
 - sound4::bigvoice::dyn::CInstance, 116
 - sound4::bigvoice::dyn::CInstance::CClient, 83
 - sound4::bigvoice::dyn::helper::CDynLoader, 100
- OutputChannels
 - sound4::bigvoice, 71
 - sound4::bigvoice::dyn, 74
- OutputSampleSize
 - sound4::bigvoice, 72
 - sound4::bigvoice::dyn, 74
- PresetChange_Kind
 - sound4::bigvoice, 70
 - sound4::bigvoice::dyn, 73
- PresetManager_InformChange
 - sound4::bigvoice::CInstance, 106

- sound4::bigvoice::dyn::CDynLib, 93
 - sound4::bigvoice::dyn::CInstance, 116
- Process creation/destruction, 24
 - bigvoice_ExitProcess, 24
 - bigvoice_FreeParameters, 25
 - bigvoice_FreeParameterValue, 25
 - bigvoice_GetParameter, 25
 - bigvoice_GetProcessChunkFrames, 25
 - bigvoice_InitProcess, 26
 - bigvoice_InitProcess2, 26
 - bigvoice_InitProcess3, 26
 - bigvoice_NewParameters, 28
 - bigvoice_SetParameter, 28
 - bigvoice_TerminateProcess, 28
- ProcessAnyAudio
 - sound4::bigvoice::CInstance, 106
 - sound4::bigvoice::dyn::CInstance, 116
- ProcessAudio
 - sound4::bigvoice::CInstance, 107
 - sound4::bigvoice::dyn::CDynLib, 93
 - sound4::bigvoice::dyn::CInstance, 117
- ProcessAudio_Planar
 - sound4::bigvoice::CInstance, 107
 - sound4::bigvoice::dyn::CDynLib, 94
 - sound4::bigvoice::dyn::CInstance, 117
- ProcessJson
 - sound4::bigvoice::CInstance::CClient, 81
 - sound4::bigvoice::dyn::CDynLib, 94
 - sound4::bigvoice::dyn::CInstance::CClient, 83
- ProcessLoop
 - sound4.bigvoice.cl-pipe-cxx.cpp, 136
 - sound4.bigvoice.cl-pipe-dyn.cpp, 140
 - sound4.bigvoice.cl-pipe.cpp, 146
- Read
 - MyPresetManager, 128
 - sound4::bigvoice::CPresetLoader, 122
 - sound4::bigvoice::dyn::CPresetLoader, 124
- Remove
 - MyPresetManager, 128
 - sound4::bigvoice::CPresetLoader, 122
 - sound4::bigvoice::dyn::CPresetLoader, 124
- Rename
 - MyPresetManager, 128
 - sound4::bigvoice::CPresetLoader, 122
 - sound4::bigvoice::dyn::CPresetLoader, 124
- S16_BE
 - Audio managing (any audio), 43
- S16_LE
 - Audio managing (any audio), 43
- S16_NATIVE
 - Audio managing (any audio), 43
- S24_BE
 - Audio managing (any audio), 43
- S24_LE
 - Audio managing (any audio), 43
- S24_NATIVE
 - Audio managing (any audio), 43
- S32_BE
 - Audio managing (any audio), 43
- S32_LE
 - Audio managing (any audio), 43
- S32_NATIVE
 - Audio managing (any audio), 43
- Sample using C interface, 15
- Sample using C++ interface wrapper, 64
- Sample using C++ interface wrapper with dynamic library loading, 65
- SAMPLE_RATE
 - sound4.bigvoice.cl-pipe.cpp, 145
- SampleFormat
 - sound4::bigvoice, 71
 - sound4::bigvoice::dyn, 73
- SampleRate
 - sound4::bigvoice, 72
 - sound4::bigvoice::dyn, 74
- SaveState
 - sound4::bigvoice::dyn::CDynLib, 94
- SetBus
 - sound4::bigvoice::CInstance, 108
 - sound4::bigvoice::dyn::CInstance, 118
- SetInstanceBus
 - sound4::bigvoice::dyn::CDynLib, 94
- SetLoggerCallback
 - sound4::bigvoice::dyn::CDynLib, 94
- SetLogSeverity
 - sound4::bigvoice::dyn::CDynLib, 94
- SetMetadata
 - sound4::bigvoice::CInstance, 108
 - sound4::bigvoice::dyn::CDynLib, 95
 - sound4::bigvoice::dyn::CInstance, 118
- SetParam
 - sound4::bigvoice::CInstance, 108
 - sound4::bigvoice::dyn::CInstance, 118
- SetParameter
 - sound4::bigvoice::dyn::CDynLib, 95
- SetPresetManager
 - sound4::bigvoice::CInstance, 109
 - sound4::bigvoice::dyn::CDynLib, 95
 - sound4::bigvoice::dyn::CInstance, 119
- SetupPath
 - MyPresetManager, 128
- sound4, 69
- sound4.bigvoice.cl-pipe-cxx.cpp, 135, 137
 - GetEnv, 135
 - main, 136
 - MyLogger, 136
 - ProcessLoop, 136
 - Usage, 136
- sound4.bigvoice.cl-pipe-dyn.cpp, 139, 141
 - GetEnv, 139
 - main, 139
 - MyLogger, 140
 - ProcessLoop, 140
 - Usage, 140
- sound4.bigvoice.cl-pipe.cpp, 144, 146

- CHANNEL_COUNT, 144
- CHUNK_SIZE, 145
- main, 145
- MyLogger, 145
- ProcessLoop, 146
- SAMPLE_RATE, 145
- Usage, 146
- sound4.bigvoice.cl.h, 149, 153
 - BIGVOICE_DECL, 152
 - BIGVOICE_HAS_CLOUDBUS, 152
 - BIGVOICE_HAS_WEBSERVER, 152
- sound4.bigvoice.cl.hpp, 156, 158
 - fs_strdup, 158
- sound4.bigvoice.cl_dyn.hpp, 165, 167
 - fs_strdup, 166
- sound4::bigvoice, 69
 - ChunkFrames, 71
 - InputChannels, 71
 - InputSampleSize, 71
 - log_cb_t, 70
 - LogSeverity, 70
 - OutputChannels, 71
 - OutputSampleSize, 72
 - PresetChange_Kind, 70
 - SampleFormat, 71
 - SampleRate, 72
- sound4::bigvoice::CBus, 78
 - ~CBus, 79
 - CBus, 78
 - Get, 79
- sound4::bigvoice::CInstance, 100
 - ~CInstance, 102
 - AddAudio, 102
 - CInstance, 101, 102
 - Create, 102
 - GetAudio, 103
 - GetBufferIn, 103
 - GetBufferOut, 103
 - GetChunkFrames, 104
 - GetEstimatedDelay, 104
 - GetMetadataInfos, 104
 - GetParam, 105
 - IsOk, 105
 - NewClient, 105
 - OnUpdateThreadStart, 105
 - OnUpdateThreadStop, 106
 - operator=, 106
 - PresetManager_InformChange, 106
 - ProcessAnyAudio, 106
 - ProcessAudio, 107
 - ProcessAudio_Planar, 107
 - SetBus, 108
 - SetMetadata, 108
 - SetParam, 108
 - SetPresetManager, 109
 - StartWebServer, 109
 - Stop, 109
 - StopWebServer, 110
 - Terminate, 110
- sound4::bigvoice::CInstance::CClient, 80
 - ~CClient, 81
 - CClient, 81
 - operator=, 81
 - ProcessJson, 81
- sound4::bigvoice::CPresetLoader, 120
 - ~CPresetLoader, 121
 - CPresetLoader, 121
 - Exists, 121
 - GetAll, 121
 - IsReadOnly, 122
 - Read, 122
 - Remove, 122
 - Rename, 122
 - Write, 122
- sound4::bigvoice::dyn, 72
 - ChunkFrames, 74
 - InputChannels, 74
 - InputSampleSize, 74
 - log_cb_t, 73
 - LogSeverity, 73
 - OutputChannels, 74
 - OutputSampleSize, 74
 - PresetChange_Kind, 73
 - SampleFormat, 73
 - SampleRate, 74
- sound4::bigvoice::dyn::CBus, 79
 - ~CBus, 80
 - CBus, 80
 - Get, 80
- sound4::bigvoice::dyn::CDynLib, 84
 - AddAudio, 87
 - AddPadAudio, 87
 - AudioConvertFrom, 87
 - AudioConvertTo, 87
 - AudioMonoFromLiveStereo, 87
 - AudioMonoToLiveStereo, 87
 - CDynLib, 86
 - DeleteClient, 88
 - ExitProcess, 88
 - FreeBus, 88
 - FreeJsonAnswer, 88
 - FreeMetadataInfos, 88
 - FreeParameters, 88
 - FreeParameterValue, 89
 - GetAudio, 89
 - GetAudioInputCount, 89
 - GetAudioOutputCount, 89
 - GetBufferIn, 89
 - GetBufferOut, 89
 - GetBytesFromFormat, 90
 - GetChannelCount, 90
 - GetChunkSizeInFrames, 90
 - GetEstimatedDelay, 90
 - GetFormatFromName, 90
 - GetFormatName, 90
 - GetMaxPacketFrame, 91

- GetMetadataInfos, 91
- GetOutputCount, 91
- GetParameter, 91
- GetPossibleChunkSizeInFrames, 91
- GetProcessChunkFrames, 91
- GetSampleRate, 92
- GetVersion, 92
- InitProcess, 92
- InitProcess2, 92
- InitProcess3, 92
- IsOk, 86
- Load, 86
- MonoToStereo, 92
- MonoToStereo_Planar, 93
- NewBus, 93
- NewClient, 93
- NewParameters, 93
- PresetManager_InformChange, 93
- ProcessAudio, 93
- ProcessAudio_Planar, 94
- ProcessJson, 94
- SaveState, 94
- SetInstanceBus, 94
- SetLoggerCallback, 94
- SetLogSeverity, 94
- SetMetadata, 95
- SetParameter, 95
- SetPresetManager, 95
- StartUpdateThread, 95
- StereoToMono, 95
- StereoToMono_Planar, 95
- StopUpdateThread, 96
- TerminateProcess, 96
- WaitUpdateThreadReady, 96
- Webserver, 96
- Webserver_Status, 96
- Webserver_Stop, 96
- Webserver_tcp, 97
- Webserver_tcp2, 97
- sound4::bigvoice::dyn::CInstance, 110
 - ~CInstance, 112
 - AddAudio, 112
 - CInstance, 112
 - Create, 112
 - GetAudio, 113
 - GetBufferIn, 113
 - GetBufferOut, 113
 - GetChunkFrames, 114
 - GetEstimatedDelay, 114
 - GetMetadataInfos, 114
 - GetParam, 115
 - IsOk, 115
 - NewClient, 115
 - OnUpdateThreadStart, 115
 - OnUpdateThreadStop, 116
 - operator=, 116
 - PresetManager_InformChange, 116
 - ProcessAnyAudio, 116
 - ProcessAudio, 117
 - ProcessAudio_Planar, 117
 - SetBus, 118
 - SetMetadata, 118
 - SetParam, 118
 - SetPresetManager, 119
 - StartWebServer, 119
 - Stop, 119
 - StopWebServer, 120
 - Terminate, 120
- sound4::bigvoice::dyn::CInstance::CClient, 82
 - ~CClient, 83
 - CClient, 83
 - operator=, 83
 - ProcessJson, 83
- sound4::bigvoice::dyn::CPresetLoader, 123
 - ~CPresetLoader, 123
 - CPresetLoader, 123
 - Exists, 123
 - GetAll, 124
 - IsReadOnly, 124
 - Read, 124
 - Remove, 124
 - Rename, 124
 - Write, 124
- sound4::bigvoice::dyn::helper, 75
 - DynFunc_t, 75
 - DynLib_t, 75
- sound4::bigvoice::dyn::helper::CDynLoader, 97
 - ~CDynLoader, 98
 - CDynLoader, 98
 - Close, 98
 - GetSymbol, 99
 - GetSymbol_safe, 99
 - GetThisLibraryPath, 99
 - IsOk, 99
 - Load, 99
 - operator bool, 100
 - operator=, 100
- sound4::bigvoice::dyn::helper::DynFuncHolder< T >, 125
 - DynFuncHolder, 125
 - IsOk, 126
 - operator bool, 126
 - operator T*, 126
- sound4::bigvoice::dyn::helper::SampleFormat< float >, 130
 - format, 130
- sound4::bigvoice::dyn::helper::SampleFormat< int16_t >, 131
 - format, 132
- sound4::bigvoice::dyn::helper::SampleFormat< int32_t >, 133
 - format, 133
- sound4::bigvoice::dyn::helper::SampleFormat< T >, 129
- sound4::bigvoice::helper, 76
- sound4::bigvoice::helper::SampleFormat< float >, 131

- format, [131](#)
- sound4::bigvoice::helper::SampleFormat< int16_t >, [132](#)
- format, [132](#)
- sound4::bigvoice::helper::SampleFormat< int32_t >, [133](#)
- format, [134](#)
- sound4::bigvoice::helper::SampleFormat< T >, [130](#)
- SOUND4_INVALID_WEBSERVER_ID
 - Web server, [60](#)
- SOUND4_WEBSERVER_HTTP_OK
 - Web server, [60](#)
- SOUND4_WEBSERVER_HTTPS_OK
 - Web server, [61](#)
- StartUpdateThread
 - sound4::bigvoice::dyn::CDynLib, [95](#)
- StartWebServer
 - sound4::bigvoice::CInstance, [109](#)
 - sound4::bigvoice::dyn::CInstance, [119](#)
- std, [76](#)
- StereoToMono
 - sound4::bigvoice::dyn::CDynLib, [95](#)
- StereoToMono_Planar
 - sound4::bigvoice::dyn::CDynLib, [95](#)
- Stop
 - sound4::bigvoice::CInstance, [109](#)
 - sound4::bigvoice::dyn::CInstance, [119](#)
- StopUpdateThread
 - sound4::bigvoice::dyn::CDynLib, [96](#)
- StopWebServer
 - sound4::bigvoice::CInstance, [110](#)
 - sound4::bigvoice::dyn::CInstance, [120](#)
- Terminate
 - sound4::bigvoice::CInstance, [110](#)
 - sound4::bigvoice::dyn::CInstance, [120](#)
- TerminateProcess
 - sound4::bigvoice::dyn::CDynLib, [96](#)
- Update Management, [38](#)
 - bigvoice_StartUpdateThread, [38](#)
 - bigvoice_StopUpdateThread, [38](#)
 - bigvoice_WaitUpdateThreadReady, [39](#)
- Usage
 - sound4.bigvoice.cl-pipe-cxx.cpp, [136](#)
 - sound4.bigvoice.cl-pipe-dyn.cpp, [140](#)
 - sound4.bigvoice.cl-pipe.cpp, [146](#)
- verbose
 - Global setup, [21](#)
- verbose2
 - Global setup, [21](#)
- verbose3
 - Global setup, [21](#)
- verbose4
 - Global setup, [21](#)
- verbose5
 - Global setup, [21](#)
- WaitUpdateThreadReady
 - sound4::bigvoice::dyn::CDynLib, [96](#)
- warning
 - Global setup, [21](#)
- Web server, [60](#)
 - bigvoice_Webserver, [61](#)
 - bigvoice_Webserver_Status, [62](#)
 - bigvoice_Webserver_Stop, [62](#)
 - bigvoice_Webserver_tcp, [63](#)
 - bigvoice_Webserver_tcp2, [63](#)
 - SOUND4_INVALID_WEBSERVER_ID, [60](#)
 - SOUND4_WEBSERVER_HTTP_OK, [60](#)
 - SOUND4_WEBSERVER_HTTPS_OK, [61](#)
- Webserver
 - sound4::bigvoice::dyn::CDynLib, [96](#)
- Webserver_Status
 - sound4::bigvoice::dyn::CDynLib, [96](#)
- Webserver_Stop
 - sound4::bigvoice::dyn::CDynLib, [96](#)
- Webserver_tcp
 - sound4::bigvoice::dyn::CDynLib, [97](#)
- Webserver_tcp2
 - sound4::bigvoice::dyn::CDynLib, [97](#)
- Write
 - MyPresetManager, [129](#)
 - sound4::bigvoice::CPresetLoader, [122](#)
 - sound4::bigvoice::dyn::CPresetLoader, [124](#)